TRABAJO FIN DE GRADO

Programación de proyecciones visuales interactivas mediante Kinect y Processing

Grado de Ingeniería electrónica industrial y automática

Javier Picazas Molinos

ÍNDICE

INTRODUCCIÓN

Inicialmente este trabajo surgió de la idea de unir tecnología y danza para crear obras que innovasen en estas áreas creando algo nuevo con cosas tan diferentes, creando un lenguaje entre ambas para comunicar y enlazar lo que se transmite con el cuerpo por medio de la danza y lo que se transmite con la tecnología por medio de la imagen.

La profesionalización de las artes escénicas fue posible gracias a la aplicación de las tecnologías de principios del siglo XIX, que posibilitaron el paso de la danza desde el acontecimiento social al hecho artístico. La introducción de la luz de gas permitió generar lo que hoy en día conocemos como dispositivo teatral a la italiana: un escenario iluminado y un patio de butacas apagado desde el que la gente observa el espectáculo. En apariencia es una organización muy sencilla, pero tiene implicaciones muy profundas en el mundo de las artes escénicas (básicamente la disociación del rol del espectador frente al del bailarín, cuando hasta ese momento en los bailes de corte todos los asistentes al hecho escénico eran partícipes y espectadores al mismo tiempo). En el siglo XXI la tecnología sigue impulsando la creación escénica, tanto en los temas que preocupan a los coreógrafos de hoy como en las características formales con las que pueden desarrollar sus creaciones. Este proyecto pretende ser una muestra de cómo el arte y la técnica pueden avanzar de manera conjunta. Para la tecnología, la danza plantea nuevos retos que pueden provocar innovaciones en el campo tecnológico. Y para la danza, la tecnología plantea nuevos temas y respuestas creativas. Existe una simbiosis entre ambas disciplinas que todavía tiene mucho recorrido y que en este proyecto se pretende esbozar.

Dentro de la Universidad podemos encontrar un contexto idóneo para plantear una investigación en torno a la danza y la ingeniería automática. Existe en la Universidad un servicio de cultura dedicado a las artes escénicas, el Aula de las Artes (www.auladelasartes.es), y el RoboticsLab del dto. de Ingeniería de Sistemas y Automática (http://roboticslab.uc3m.es/roboticslab). La colaboración entre ambos ha permitido el desarrollo de esta investigación y su labor puede perpetuar este tipo de dinámicas que contribuyan al desarrollo de proyecto híbridos de interés en ambos campos.

OBJETIVOS

Los objetivos pensados para este trabajo estén relacionados con la danza, la tecnología y la unión de ambos, no centrándose en uno solo de estos dos campos. Estos son:

- Generar un diálogo entre las disciplinas técnica y dancística, de manera que ambas se retroalimenten en un proceso creativo que incluya la investigación y la búsqueda de nuevos lenguajes para aumentar la poética de la escena.
- Crear una plataforma de trabajo para el aula de las artes de la UC3M ya sea para imprevistos, ensayos, obras, etc.
- Componer un pequeño espectáculo con los resultados de la puesta en común.
- Ser capaces de establecer puentes comunicativos entre dos ámbitos del saber en apariencia muy distantes como la Ingeniería y las Humanidades, rompiendo barreras para que futuras colaboraciones sean posibles en ambos campos dentro del contexto de la universidad.
- Comprender las dinámicas necesarias para desarrollar un trabajo profesional como consultor tecnológico de una compañía de danza, incluyendo los aspectos organizativos (preproducción y gestión del montaje de un espectáculo) y los aspectos artísticos (cómo apoyar las peticiones técnicas del cliente y búsqueda de alternativas para problemas no solventables).

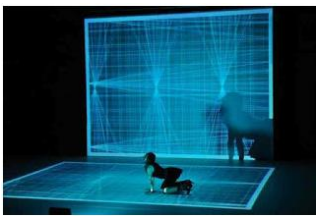
ESTADO-DEL-ARTE

Dentro del mundo del arte uno de los grandes objetivos que han aparecido en los últimos años ha sido la unión de la danza y la tecnología, ya fuese utilizando sensores para variar la música con el movimiento del bailarín o con proyecciones sobre este creando un mundo virtual en el que todo es posible.

El primer contacto que hubo con la danza respecto a proyecciones, que es el campo que nosotros utilizamos para este trabajo, fue por medio de vídeos. El bailarín se preparaba una coreografía que se llevaba a cabo a la vez que el video coordinándose o proyectando un video desconocido por el bailarín para que él improvisara a partir de este video.

Más adelante nuevos grupos de artistas quisieron avanzar en este campo buscando algo que no solo se proyectase sobre un plano y a partir de allí uno decidía que hacer, sino algo que no solo se proyectase, algo que se coordinase con los movimientos del bailarín cambiando ese mundo creado a su antojo con su cuerpo. De esa idea surgieron obras como "Avatar" de M.Angeles Angulo y Román Torre, una obra en la que se capta el movimiento del bailarín por medio de una cámara y a partir de esos datos obtenidos el mundo que se ha creado va variando con sus movimientos.

Los programas también han ido avanzando. Inicialmente se proyectaban figuras en dos dimensiones con un único proyector y actualmente se puede proyectar en tres dimensiones con varios proyectores o siguiendo una figura gracias a la imagen de la cámara. Un ejemplo de proyección en dos planos es la imagen de la derecha.



Otra de las obras conocidas que utilizan proyecciones es "Stocos" de Pablo Palacio y Muriel Romero, la cual ha sido representada en varios festivales y cierra una trilogía que conecta la danza contemporánea y las nuevas tecnologías. Su objetivo es crear dependencias de comportamiento y relaciones estéricas entre los bailarines, la música y las proyecciones que se generan sobre estos.

Aunque son pocas las compañías que trabajan este campo el avance va siendo cada vez mayor combinando música, danza e imagen. Para un futuro esta unión de arte y tecnología ira creciendo, ya que a medida que avanza la tecnología se disponen de más medios y de más opciones para utilizar dentro de una obra artística. El avance más claro va a ser el paso de la kinect 1 a la Kinect 2 ya que como se muestra en el apartado "Trabajos futuros" aumenta su calidad en todos los aspectos.

ESTRUCTURA DEL DOCUMENTO

La estructura del documento está compuesta por 9 puntos principales con un tema diferente en cada uno y estos a su vez divididos en puntos. En esta estructura se ha separado la parte de programación con la parte artística, ya que aunque se unan para crear algo nuevo dentro del arte son mundos muy diferentes, de ahí la dificultad de comunicarlos y finalmente unirlos. Estos 9 puntos son:

1. Introducción: Para este apartado se ha creado un pequeño texto que introduce al trabajo explicando el porqué de dicho trabajo y cómo comenzó. Es este apartado también se explicarán los objetivos del trabajo, la estructura del documento, la sinestesia, la metáfora cuerpo-proyección y la obra "Giselle".

2. Plataforma de trabajo: En este apartado se tratará el sensor kinect, los programas utilizados para llevar a cabo el trabajo como Processing, Github, etc.

3. Reuniones y ensayos con el laboratorio de danza: se tratarán todas las reuniones llevadas a cabo durante el año académico y los conceptos y las conclusiones de que se han conseguido por medio de estas.

4. Implementación de escenarios interactivos: Comienza con una explicación de la implementación general del sistema para luego pasar a los escenarios que se han creado de manera independiente. Se divide en "Elementos estéticos", donde se explica estéticamente como es cada uno de los escenarios, y "Control de los elementos estéticos", donde se explica cómo se ha llegado a crear lo explicado en el apartado anterior por medio del código.

5. Puesta en escena: Se trata la dramaturgia y su integración dentro de estos, como han salido los programas durante la danza.

6. Conclusiones: Aparecen las conclusiones del trabajo fin de grado.

7. Trabajos futuros: A partir de lo hecho en este trabajo, se nombran algunas de las mejoras que se podrían realizar sobre este o algunos trabajos que se podrían conseguir cogiendo este como base, como por ejemplo, introducir la cámara Kinect 2 en vez de la Kinect 1.

8. Bibliografía: Es una recopilación de todos los lugares donde se ha obtenido información para la creación de este trabajo, está compuesta por los libros leídos, las páginas web utilizadas, etc.

9. Finalmente se añaden una serie de anexos en los que se enseñan fotos de los ensayos, el cartel del Óxido Fest y el código del programa que se expondrá en el Óxido Fest.

OBRA "GISELLE"

Giselle es la pieza del repertorio de ballet clásico más antigua que se conserva. La danza, al ser un arte efímero, no conserva un patrimonio anterior a los ballets románticos decimonónicos. Sólo ha sido posible que esta obra llegue hasta nuestros días gracias a que ha sido representada de forma ininterrumpida desde su fecha de estreno en 1841 hasta la actualidad. Este dato ya convierte esta pieza en susceptible de cualquier revisión, por su relevancia en la historia de la danza y por ser síntesis expresiva de un periodo artístico. Pero cabe apuntar que fue una de las primeras obras en aplicar las tecnologías a la escena: la luz de gas y las zapatillas de puntas (una prótesis que sirve a las bailarinas para sostener el peso de su cuerpo de manera artificial en las puntas de los dedos de sus pies). La combinación de ambas técnicas otorgaban un aspecto fantasmagórico a las bailarinas, de forma que se amplificaba el concepto presentado en el argumento (las protagonistas de la obra son espíritus de mujeres que han muerto de desamor). Es decir, que es un buen ejemplo de cómo la tecnología fue un potenciador de la poética de la escena, no un mero catálogo de posibilidades técnicas sin valor en el discurso artístico. Probablemente fue la primera vez que esta comunión tuvo resultados brillantes, por lo que aplicar de nuevo la simbiosis de la danza y la ingeniería es realmente muy pertinente.

Además de lo anteriormente expuesto, "Giselle" es uno de los mitos fundacionales de la danza clásica que está siendo trabajado como tal por el Laboratorio de Danza, la compañía de danza contemporánea del Aula de las Artes de la Universidad en el contexto del proyecto europeo de cultura Crossing Stages (www.crossingstages.eu), que plantea una relectura del mito clásico desde el pensamiento y la escena contemporánea. De esta forma, los resultados de este proyecto tendrán difusión a nivel internacional y el proyecto europeo podrá nutrirte de este diálogo que aporta interés al contenido de su programa.

Utilizando su romántico ingenio, dio forma a una obra compuesta de dos actos en los que cuentan la historia de Giselle, una aldeana casi adolescente, ingenua, apasionada por la danza, que vivía con su madre en un lugar de Alemania muy cerca del Rin que se enamora de un noble llamado Albrecht de Silesia del cual desconoce su situación social. Una vez que el guardabosques Hilarion de desvela el misterio a Giselle se produce una dramática escena que termina en locura y muerte de la heroína. Una vez que ha muerto, en medio del bosque y a la luz de la luna, en su tumba aparecen las Willis (espíritus de novias engañadas) para incorporar a su cohorte a Giselle, como desea Myrtha, su reina. Pero se impone el gran amor de Giselle hacia Albrecht y al alba las Willis desaparecen después de haber matado a Hilarión y Giselle se despide de Albrecht en romántico final.

Esta obra es una mezcla de humanidad y sobrenaturalidad uniendo amor, tragedia, locura, muerte e inmaterialidad en una muestra romántica a todas luces. Aunque la obra es del año 1841, su mensaje pertenece a todas las épocas: el que nos subraya que el amor es más fuerte que el odio.

ACTO I

La trama se desarrolla en una pequeña aldea de Alemania durante la celebración de la fiesta de la vendimia. Giselle, joven e inocente campesina, está enamorada y es correspondida por Albrecht, un noble que se ha disfrazado de aldeano para obtener su amor. El guardabosque Hilarión también ama a Giselle, quien lo rechaza confesándole que su corazón ya tiene dueño. El desdeñado pretendiente, celoso, jura vengarse. Giselle baila en la fiesta, a pesar de las recomendaciones de su madre que teme por su delicada salud. Llega el séquito del Duque de Courland con su hija Bathilde, prometida oficial de Albrecht, la que atraída por el candor y belleza de Giselle le regala su collar. Pero Hilarión, al descubrir escondida entre la maleza la espada de Albrecht, la que delata su condición de noble, aprovecha la presencia de] Duque y de su hija y lo desenmascara ante todos. Giselle, desesperada, pierde la razón y falla su débil corazón incapaz de resistir tanto dolor, cayendo muerta ante la consternación de los presentes.

ACTO II

En un bosque, al borde de una laguna, se encuentra la tumba de Giselle. A medianoche las Willis, espíritus de las novias abandonadas por sus prometidos, empiezan a bailar, y su Reina Myrtha recibe a Giselle en este mundo fantasmal. Llega Hilarión, abrumado por el remordimiento, pues se considera responsable de la muerte de la joven, y se arrodilla ante su tumba. Pero las Willis lo envuelven y lo obligan a arrojarse al lago. También Albrecht visita el sepulcro de Giselle e implora perdón por el engaño. La doncella se conmueve ante su dolor, pero la inflexible Reina de las Willis ordena a Giselle atraerlo a una danza que acabará con su vida. La joven, tratando de salvar a Albrecht, le indica que se refugie en su tumba para así destruir el sortilegio. Amanece, las Willis desaparecen, y Giselle vuelve a su sepultura. Albrecht trata de retenerla, pero la imagen de la amada se desvanece con la luz del naciente día.

PRECEPTOS TEÓRICOS

La propuesta del proyecto es generar unos visuales interactivos. Es decir, la creación escénica se basa en la proyección de imágenes que interactúan a tiempo real con el bailarín gracias al sensor kinect. Para comprender la relevancia de la proyección en este aspecto es necesario entender los términos de sinestesia y metáfora, que configuran dos pilares fundamentales de la introducción de estas tecnologías en la escena.

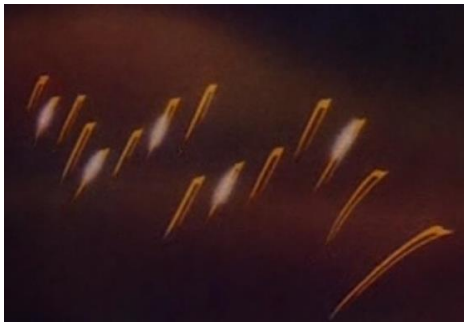
SINESTESIA

La palabra sinestesia procede del griego, que significa "unión de sensaciones". Esto lo que hace es mezclar los sentidos, haciendo que las personas que son sinestésicas puedan por ejemplo ver los colores mientras que escuchas una canción o apreciar sabores cuando alguien te habla, pudiendo ser estas interacciones de lo más variado. En estos casos no hace falta que estén involucrados solamente dos sentidos, puede haber más (por ejemplo, asociar el número 5 con el color rojo, pero al mismo tiempo con una textura lisa o rugosa), y todo ello de forma involuntaria, permitiendo jugar con el espacio o con las formas geométricas.

Este fenómeno podría darse debido a las conexiones neuronales de la persona que la posee, creando ciertas uniones entre distintas áreas en un momento poco común. Dentro de una misma persona estas asociaciones, de por ejemplo algún sonido con algún color en concreto, se mantienen estables a lo largo del tiempo en una misma persona, pero variando entre las personas. Además esta asociación es unidireccional, por ejemplo, el color rojo evoca sabor dulce, pero no por ello el sabor dulce va a evocar el color rojo.

Hay multitud de casos de sinestesia, como los que son provocados por el sabor, el olor, el dolor y hasta por las personas. Este último se refiere a que las personas por sí mismas pueden ser un estímulo para los sinestetas que las perciben de algún color o tono en particular, como si dichas personas poseyeran "auras".

Estas personas que poseen sinestesia poseen además una mayor capacidad retentiva a la hora de retener una secuencia de colores o sonidos, aunque también pueden ser peores del resto en matemáticas dado que tienden a agrupas los números por colores.



Un acercamiento a esta sensación para la gente de a pie puede ser el comienzo de la película *Fantasía*, de Disney, un buen ejemplo de simulacro de sinestesia musical. Cuando suenan instrumentos mientras aparecen figuras y líneas dependiendo de qué instrumento suene y se mueven al compás de la música acompañándola con brillos y luces.

SINESTESIA Y EL ARTE

La característica esencial de la música es despertar sentimientos en las personas, ya sea activando recuerdos o asociaciones, o simplemente de manera directa

Dentro del arte la sinestesia que más interesa es aquella que asocia la música que se escucha con colores y formas, ya que se podría plasmar por medio de pintura y animaciones la música que se escucha en ese momento. Uno de estos ejemplos es "Composición 8" de Kandinsky, que eran en realidad pinturas sinfónicas.

Dentro de este campo, en relación con la escritura mayormente, la sinestesia se utiliza como figura retórica que mezcla sensaciones auditivas, visuales, gustativas, olfativas y táctiles y además asocia elementos procedentes de los sentidos físicos con los sentimientos. Una sintesia es de primer grado cuando se asocian dos sentidos corporales diferentes y es de segundo grado si se asocian un sentido del cuerpo a una emoción, objeto o idea.

Este sería también el objetivo de las proyecciones en tiempo real, trabajar con la música que suena y el movimiento que se lleva a cabo mientras que se muestra de manera visual una serie de polígonos, formas, colones, líneas, etc. pudiendo asociar estos tres campos y unirlos en una única obra.

Dos casos de esta sinestesia en el arte serían los compositores Olivier Messiaen y Alexander Scriabin. Ellos sinestésicos, y sus obras son el ejemplo más claro del tipo de sinestesia de la que hablamos, no sólo mostrando colores a partir de la música, también en algunos casos haciendo música a partir de la escala cromática. Ellos ven colores cuando escuchan música y cada acorde es un color para ellos, todo ello asociado para dar una unión de imagen y sonido perfecta.

Un gran avance sería poder generar eso en una obra de manera que todo el mundo pudiese experimentar dicha experiencia por medio de la vista y el odio, y esto es posible gracias a la proyección en tiempo real.

METÁFORA CUERPO-PROYECCIÓN

La Teoría de la Metáfora formulada por Mark Johnson (1987), una de las teorías filosófico-cognitivas que más se ha aplicado a la música en los últimos años, concede al cuerpo un papel fundamental en la cognición. Sostiene que parte de nuestra forma de entender el mundo es metafórica en cuanto a que implica proyectar patrones de un dominio cognitivo a otro. Proyectamos "esquemas encarnados" que son "patrones recurrentes de nuestras interacciones perceptuales y programas motores".

La filosofía de la mente desde un punto de vista cientificista se desarrolló en el siglo XX por filósofos de Viena sosteniendo que la mente y el cuerpo están constituidos por un mismo tipo de material y no son dos entidades diferentes. En 1977 la mente empieza a coger un punto de interés que ha de abordarse desde las ciencias cognitivas desde todas las perspectivas posibles. A partir de entonces multitud de investigadores empiezan a investigar dentro de este campo y surgen multitud de estudios sobre ello.

La teoría de la metáfora sostiene que parte de nuestro pensamiento, nuestra forma de entender el mundo, es metafórica. Para el pensamiento abstracto es necesario utilizar esquemas (esquemas encarnados) que utilizamos para dar sentido a nuestras experiencias mediante proyecciones metafóricas, estos esquemas se forman a partir de experiencias corporales que experimenta el individuo.

Las proyecciones metafóricas son proyecciones de un esquema encarnado que están determinadas por la estructura de estos. Por ejemplo, podemos utilizar el esquema camino para comprender la trayectoria profesional de alguien. Pero además, podemos afirmar que en la personalidad de dicha persona se trasluce su trayectoria profesional.

En relación con la cultura, esta tiene cierta importancia ya que regula las experiencias corporales necesarias para los esquemas, ya sea viviendo las experiencias por medio de esta o aprendiéndolas culturalmente por empatía, y condiciona nuestras percepciones y nuestra forma de relacionarnos con los objetos. Un ejemplo sería con Superman ya que nadie ha volado nunca como Superman. Sin embargo, es muy posible que sí haya experimentado la sensación de volar, bien sea en avión, en parapente, en ala delta, en sueños, etc. De la visualización de una película en la que veamos a Superman, podemos generar determinados esquemas, sin necesidad de tener la experiencia directa del vuelo, dicho esquema es cultural.

Con relación a la proyección en el cuerpo sigue siendo igual de válida, ya que con el movimiento del cuerpo se puede generar en las personas que están viendo un espectáculo un esquema u otro sin tener que utilizar la experiencia en sí dentro de la obra. De igual modo se podría utilizar proyecciones y música con el mismo, situando en un escenario con una situación en concreto u otra utilizando por ejemplo escenas luminosas con formas suaves para generar paz en el espectador o alegría.

PLATAFORMA DE TRABAJO

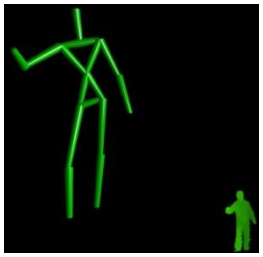
Para el desarrollo de este trabajo se ha utilizado una serie de programas y herramientas necesarias para el almacenamiento de datos, la captura de imágenes, la programación de las artes visuales y para la transformación de los datos obtenidos por la cámara a los datos del programa. Dichos programas y herramientas se describen brevemente a continuación.

SENSOR KINECT

Kinect es una cámara creada por Microsoft que posee tres tipos de cámara distintas, un motor que mueve la inclinación con la que enfoca la cámara, pudiéndolo cambiar en cualquier momento con Processing, con la librería OpenKinect, y un micrófono que capta todos los sonidos que hay a su alrededor calculando el ángulo desde donde vienen, sabiendo en que zona de la habitación se encuentra y eliminando el ruido de fondo.

Respecto a las cámaras, tienen una resolución de 640x480 pixeles y posee dos cámaras. Una cámara a color (cámara RGB) que captura la imagen de lo que está captando y otra cámara que es de profundidad que funciona con infrarrojos. Esta segunda coge la distancia a la que se encuentran todos los pixeles que captura de la imagen, siendo su rango de 0.5 a 4 metros más o menos. Aunque no es tan exacta como la última versión, la kinect 2.0, esta cámara nos permite trabajar con objetos en 3D gracias a los datos que nos aporta cada pixel funcionando con un proyector infrarrojo y una cámara IR, que es un sensor especialmente diseñado para capturar luz infrarroja, que recoge la información de los rayos infrarrojos lanzados y gracias a dicha información sabe la distancia a ese punto. No es 100% exacta ya que con objetos curvos, de cristal o con objetos que poseen una pequeña distancia entre ellos no captura bien la imagen y deja partes sin información o une dichos objetos.

También permite diferenciar usuarios (hasta seis usuarios diferentes) y poder representar un esqueleto del usuario mostrando manos, codos, hombros, cuello, cabeza, cintura, rodillas y pies, dándote la información de donde se encuentran, coordenadas (x,y,z). En la imagen se puede observar como con dichos puntos se ha creado un muñeco uniendo los puntos por cilindros verdes.



PROYECTOR

El proyector utilizado para proyectar sobre las personas todos los escenarios realizados es un proyector Epson EB-S11H, con una resolución nativa de 800x600 pixeles en formato 4:3, posee 2600lúmenes, que en estado económico se bajan a 2080, tecnología 3LCD y contraste 3000:1.

Posee dos entradas VGA y una salida VGA para ordenadores, un puerto S-Vídeo para conexiones audiovisuales tradicionales y dos puertos USB, un tipo A para proyectar imágenes JPEG a través de una memoria y otro tipo B para actualizar el software o recaudar información del proyector. La lámpara tiene una vida de 4000 a 5000 horas de duración útil.

A continuación se adjunta una tabla con las características de Epson del proyector:

SPECIFICATIONS
EB-S11H/X11H/X14H/X15

EPSON
EXCEED YOUR VISION

MODEL NUMBER		EB-S11H	EB-X11H	EB-X14H	EB-X15
Projection Technology		RGB liquid crystal shutter projection system (3LCD)			
Specifications of Main Parts					
LCD	Size	0.55" without MLA (D7)	0.55" without MLA (D8)	0.55" with MLA (D8)	
	Native Resolution	SVGA	XGA		
Projection Lens	Type	No Optical Zoom / Focus (Manual)	Optical Zoom (Manual) / Focus (Manual)		
	F-Number	1.44	1.58 - 1.72		
	Focal Length	16.7mm	16.9mm-20.28mm		
	Zoom Ratio	1-1.35 (Digital Zoom)	1 - 1.2		
	Throw Ratio	1.45 - 1.96 (Wide - Tele)	1.48 - 1.77 (Wide - Tele)		
Lamp	Type	200W UHE (E-TORL)			
	Life (Normal / Eco)	4,000 hours / 5,000 hours			
Screen Size (Projected Distance)					
Zoom: Wide		30" - 350" (0.88m - 10.44m)	30" - 300" (0.9m - 9.0m)		
Zoom: Tele		23" - 260" (0.88m - 10.44m)	30" - 300" (1.03m - 10.8m)		
Standard size		60" screen 1.77m - 2.4m	60" screen 1.8m - 2.17m		
Brightness					
White Light Output (Normal / Eco)		2,600lm / 2,080lm		3,000lm / 2,400lm	
Colour Light Output		2,600lm		3,000lm	
Contrast Ratio		3,000:1			
Internal Speaker(s)					
Sound Output		2W monoaural			
Keystone Correction					
Vertical / Horizontal		±30° / ±30°			
Auto Keystone Correction		N/A			Yes (Vertical only)
Connectivity					
Analog Input	D-Sub 15pin	2 (Blue)			
	Component	D-sub 15pin Blue molding (in common with Analog RGB connector)			
	Composite	RCA (Yellow) x 1			
	S-Video	Mini DIN x 1			
Digital Input	HDMI	N/A			1
Output Terminal	D-Sub 15pin	1 (Black)			N/A
Others	USB Type B	1			
Control I/O	RS-232C	D-sub 9pin x 1			
	USB	USB Type B x 1 (3-in-1 USB Display)			
Operating Temperature		5°C - 35°C (41°F - 95°F) (20% - 80% Humidity)			
Operating Altitude		0m - 2,286m (0ft - 7,500ft) (over 1,500m / 4,921ft with High altitude mode)			
Direct Power On / Off		Yes			
Start-Up Period		about 5 seconds, Warm-up period: 30 seconds			
Cool Down Period		Instant off or 0 second			
Power Supply Voltage		100 - 240 V AC +/- 10%, 50/60 Hz			
Power Consumption (220 - 240V)					
Lamp On (Normal / Eco)		270W / 223W			
Standby (Network On / Off)		3.3W / 0.47W			
Dimension Excluding Feet (D x W x H)		228mm x 295mm x 77mm			
Weight		Approx. 5.1lbs. / 2.3kg			
Fan Noise (Normal / Eco)		37 dB / 29dB			

ManualsLib.com

Supplied accessories
Power Cord: 1.8m
Computer Cable: 1.8m D-sub 15pin (Male) - D-sub 15pin (Male)
USB Cable: 1.8m, USB A / USB B
Remote Control: Compact type
Battery: Manganese dry cell x 2
Soft Carrying Case: ELPKS63
User's Manual Set

Optional accessories
Replacement Lamp: ELPLP67
Air Filter Set: ELPAF32

EB-S11H/EB-X11H/EB-X14H

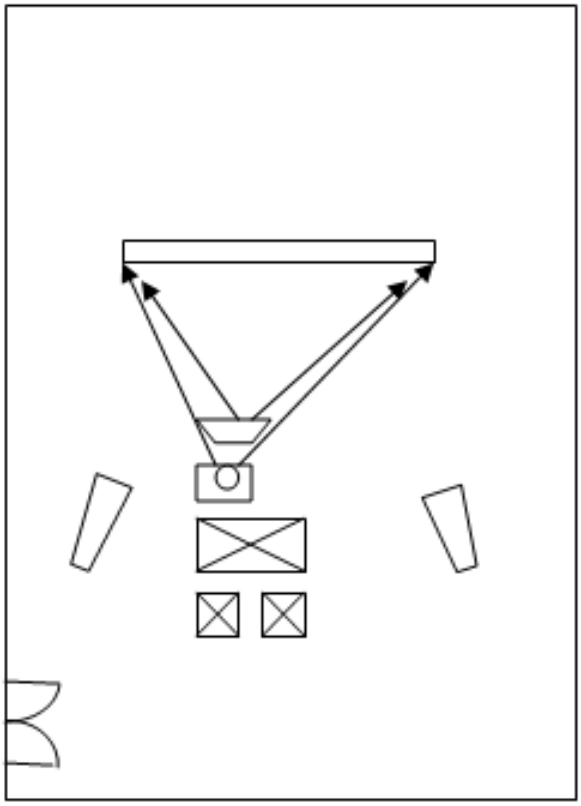
EB-X15

EPSON and EXCEED YOUR VISION are registered trademarks of Seiko Epson Corporation.
All other product names and other company names used herein are for identification purposes only and are the trademarks or registered trademarks of their respective owners.
EPSON disclaims any and all rights in those marks. Projected images shown herein are simulations. The actual product design and contents may vary. Specifications are subject to change without notice.

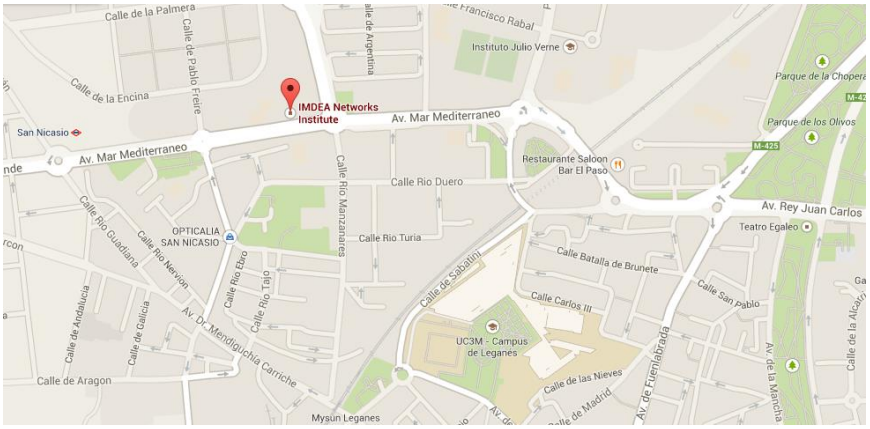
LABORATORIO DE DANZA

El espacio utilizado por el laboratorio de danza es una amplia sala con suelo especial para baile en la cual se puede colgar del techo una gran tela blanca que hace de pantalla para nuestro proyector. Posee también cortinas negras para las ventanas, las cuales dejan la sala totalmente a oscuras mejorando la proyección, además de focos apuntando al centro de la sala con los que se regula la cantidad de luz que se quiere. Un plano rápido de la sala sería el siguiente:

- Silla.
- Mesa.
- Proyector.
- Kinect.
- Foco.
- Pantalla.



Dicho aula se encuentra en el edificio de la Universidad Carlos III de Madrid que se encuentra en Leganés en la Avenida Mediterráneo, cerca de la parada de metro de San Nicasio.



PROCESSING

Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java (aunque es más sencillo que este) y que es utilizado para proyectos multimedia y diseño digital. Inicialmente fue creado para enseñar fundamentos de la programación en el contexto visual, una alfabetización de software dentro de las artes visuales y la cultura visual dentro de la tecnología, y a día de hoy lo utilizan todo tipo de personas para hacer programas con 2D, visualización de datos, la composición musical, la creación de redes, la exportación de archivos en 3D, etc. La fundación fue creada en primavera del año 2001 por Ben Fry y Casey Reas y su manera de recaudación es por medio de donaciones que hace la gente o las empresas. Sus trabajadores son voluntarios, ya que con las donaciones la fundación no gana suficiente para contratar a alguien y utilizan el dinero recaudado para mejorar Processing y poder crear nuevas librerías para este.

Hay dos maneras de añadir una librería a Processing. La más sencilla es en el mismo programa accediendo a la pestaña Sketch > Import Library… seleccionando la opción "Add Library…", de ese modo sale una lista con todas las librerías que se encuentran en la web de Processing. Una vez encontrada solamente se selecciona y el programa se instala solo. El único requisito para esta manera es que la librería se encuentre en la base de datos de Processing, sino no aparecerá. La segunda manera es descargando los archivos de la librería y guardándolos en la carpeta creada por Processing, en mi caso es C:\Users\Genral\Documents\Processing\libraries. Hay algunas librerías que no pueden estar instaladas a la vez, como es el caso de la librería "física" y "Box2D", por lo que dependiendo de cuál utilicemos habrá que estar borrando una y pegando otra en esta carpeta.

Processing lleva una cantidad de 213 versiones, siendo la más nueva la versión "Processing 2.1.1". Nosotros trabajamos con la versión anterior, la versión 2.1. El mayor salto fue a la versión 2.0 en la cual se añadió un procesamiento de gráficos más rápido, una nueva infraestructura para trabajar con los datos, poder programar en Android y JavaScript, mejoras de video y captura de imágenes y la incorporación de una OPENGL moderna.

Está disponible gratuitamente para Linux, Mac y Windows y se puede complementar con más de 100 librerías diferentes que facilitan el trabajo, la mayoría se encuentran en su página web. Las versiones utilizadas en el trabajo son la versión para Windows de 32 bits en el puesto de trabajo diario y la versión para Mac en el puesto utilizado para las muestras al aula de las artes.

Para llevar a cabo este trabajo se ha debido de complementar con los programas OpenNI y NITE, ya que sin ellos no se puede trabajar con kinect en Processing. También se ha utilizado la librería SimpleOpenNI, que es una librería especializada en la toma de datos de la kinect gracias a los programas nombrados anteriormente y el funcionamiento con estos en Processing.

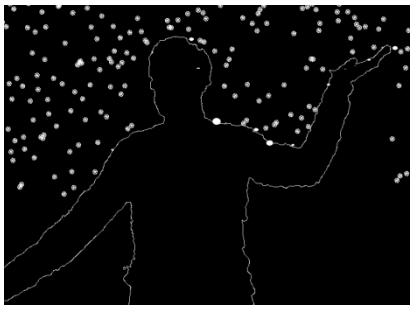
Dentro de Processing, podemos encontrar gran cantidad de librerías que poseen cada una sus características enfocadas a un tema. Estas librerías facilitan mucho la programación en Processing, ya que permiten hacer cosas de forma muy sencilla con pocos comandos en vez de tener que programar código complejo. Se pueden encontrar librerías con funciones 3D, de animación, para uso de datos externos al programa como imágenes, de sonido, para formas geométricas, de hardware, de video, etc.

Las librerías utilizadas en este trabajo son: SimpleOpenNI, BlobDetection y Box2D. Dichas librerías se explicarán a continuación.

BLOBDETECTION

La librería BlobDetection es una librería que se utiliza para generar una silueta que rodea "manchas" en una imagen. Calcula pixel por pixel detectando aquellos que poseen una gran diferencia de color entre ellos y va generando un borde con todos aquellos puntos que rodea la forma. También tiene la capacidad de generar un rectángulo que acote el espacio que ocupa dicha forma, aunque esta segunda capacidad no ha sido utilizada en este trabajo, ya que esta misma función ha sido creada por mí.

Esta función se ha utilizado en las imágenes que se capturaban con la kinect para dibujar la silueta de la persona y así poder utilizarla con el escenario que se haya creado, un ejemplo sería un escenario que va cayendo nieve y al entrar en contacto con el cuerpo se funde. Dicha silueta se ha generado a partir de la cámara depth de la kinect y por ello tiene alguna sección un poco irregular.



La parte más positiva de esta librería es que coge la silueta de todas las cosas que aparezcan en la imagen, por lo que si utilizas la cámara depth de la kinect (de profundidad) cogerá la silueta de todo lo que aparezca y no solo de la persona, como hace SimpleOpenNI con la cámara User.

BOX2D

Box2D es una librería de Processing creada por Erin Catto, la cual permite la simulación en 2D de cuerpos rígidos. Permite la creación de cuerpos rígidos con la forma y los parámetros que se desee y su interactuación con el escenario que se haya creado con otros cuerpos, estos cuerpos pueden ser estáticos (se utilizan para generar el escenario deseado ya que no se pueden mover), cinemáticos (Cuerpos que se mueven con una velocidad dada pero no responden a fuerzas que se les ejerzan) o dinámicos. También se tiene que crear un mundo donde se puede generar fuerzas como la gravedad con un valor y una dirección constante o fuerzas que dependan de otros valores como la distancia a un punto o el ángulo que forme. Este mundo se debe de actualizar cada vez que se lleva a cabo la función "void draw ()" con el código "box2d.step ();", sino no funcionará el programa.

Estos cuerpos pueden interactuar entre ellos con las fuerzas, ya que también permite el cálculo de las colisiones dependiendo de los parámetros que hayas dado a cada cuerpo. Todos estos cuerpos tienen una forma en 2D, son duros como diamantes y gracias a las funciones que forman parte de la clase que se crea con el cuerpo se puede añadir características como:

- GravityScale: Se utiliza para dar una gravedad a un cuerpo en concreto, diferente a la gravedad del mundo creado.
- Shape: Es la forma que se le da al cuerpo. Puede ser un cuadrado una circunferencia o alguna forma que hayamos creado nosotros, un polígono.
- Density: Añade una densidad al cuerpo, es una variable tipo float que puede ser cero o positiva. Se encarga de la estabilidad en el apilado de cuerpos.
- Friction: Añade un coeficiente de rozamiento al cuerpo, lo cual genera que cuando dos cuerpos caigan o se muevan juntos roten. Como en la densidad, es una variable tipo float que suele valer 0.3-0.4 y debe de estar entre cero y uno, siendo uno la fricción más fuerte y cero eliminándola.
- Restitution: Es el parámetro que se encarga de las colisiones, dependiendo de su valor los cuerpos rebotarán al chocar contra otros cuerpos o tendrán una colisión no elástica. Su valor está entre cero y uno, siendo uno una colisión elástica y cero una colisión no elástica.

En los escenarios utilizados se ha utilizado para aquellos escenarios en los que se ha utilizado un cuerpo no muy complejo y se le ha ido ejerciendo fuerzas a los cuerpos dinámicos en función de sus parámetros y utilizando cuerpos estáticos para acotar la zona y evitar así que los cuerpos se saliesen del espacio programado. También se han creado escenarios que utilizaban ambos tipos de cuerpos pero con el uso del ratón, por lo que no ha sido posible su uso en el trabajo.

En el ejemplo de la imagen se ha utilizado Box2D para crear un mundo en el que existen unos cuerpos que son circunferencias negras que son atraídas por el centro de masas del usuario hasta que este estira los brazos y las partículas salen despedidas en todas direcciones.

SIMPLEOPENNI

Simple-OpenNI es una librería de Processing que se encarga de recibir la información de OpenNI y NITE para poder trabajar con ella en Processing. El primer paso, como con todas las librerías es importarla con el código "import SimpleOpenNI.*;" y "SimpleOpenNI kinect;". Después en la función "void setup()" es donde se activan las cámaras que se va a utilizar en el programa, utilizando los códigos:

- "kinect.enableDepth ();": Habilita la cámara de profundidad.
- "kinect.enableUser ();": Habilita la función que distingue a cada usuario y le crea una máscara a cada uno.
- "kinect.enableRGB ();": Habilita la cámara RGB que muestra la imagen tal y como la ve el ojo humano.
- "kinect.setMirror (true/false);": Habilita la función espejo, con la cual Processing le da la vuelta a la imagen que recibe de kinect para que así parezca un espejo cuando la vemos en el monitor del ordenador.

Una vez se han habilitado todas las funciones se pasa a la función "void draw()", en la cual es obligatorio actualizar la información recibida de la kinect con "kinect.update(); ", ya que si no se hace el programa no funcionará. Una vez actualizados los datos de la kinect, se pueden utilizar la multitud de funciones que ofrece. Las utilizadas en el trabajo son:

- Calcular el centro de masas, con la función "User" activada se puede calcular las coordenadas del centro de masas de cada usuario y copiarlas en un vector con el código:

```
for (int i=0; i<userList.size(); i++){
  int userId = userList.get(i); //getting user data
  kinect.getCoM(userId, position);
  kinect.convertRealWorldToProjective(position, position);

  jointPos[i][0] = int(position.x*reScale);
  jointPos[i][1] = int(position.y*reScale);
}
```

- calcular la distancia entre el pixel seleccionado y la kinect por medio de la cámara de profundidad con el código:

```
for(int x = 0; x < 640; x++){           //See all the pixels
  for(int y = 0; y < 480; y++){
    clickPosition = x + (y*640);        //We see which pixel we are working on
    clickedDepth = depthValues[clickPosition];    //See the pixel's value
    if (clickedDepth > 455){
    if (maxValue > clickedDepth){
      cam.pixels[ clickPosition] = color(0);}
    }
  }
}
```

- Saber si un pixel pertenece a la máscara de algún usuario y en ese caso pintarlo de algún color con el código:

```
if (kinect.getNumberOfUsers() > 0) {
  for (int z=0; z<userList.length; z++){
    for(int h = 0; h < 480; h++){            //See all the pixels
     for(int w = 0; w < 640; w++){
       clickPosition = w + (h*640);         //We see which pixel we are working on
       if (userMap[clickPosition] != 0) {    //if It's a user's pixel its value is 1
            cam.pixels[ clickPosition] = color(0, 200, 0);
         }
       }
    }
```

- Calcular las coordenadas de los puntos del esqueleto del usuario, dependiendo de que parte de este te interese. Un ejemplo del código para calcular el punto en el que se encuentra el cuello sería:

```
PVector jointPos = new PVector(0,0,0);
 for(int i=0;i<userList.length;i++)
{
  int userId = userList [i];
 kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_NECK,jointPos);
 println("Neck:"+jointPos);
}
```

De todas las funciones que han aparecido, la única que finalmente no se ha utilizado, o se ha utilizado muy poco, ha sido el esqueleto, debido a que se debe de esperar un tiempo hasta que el programa calcula el esqueleto del usuario y porque es muy inestable respecto a otras funciones y se pierden los parámetros de vez en cuando o con algunos movimientos como dar vueltas. El uso más utilizado en algunos escenarios era para saber en qué posición se encontraban las manos, para así con esa información pasar de un escenario a otro cuando las manos pasaban algún umbral.

OPENNI

Es un driver necesario para el uso de NITE y para el funcionamiento de la kinect con Processing y SimpleOpenNI. Es un Framework de código abierto que permite manejar la kinect en un ordenador comunicándose con los sensores de audio, video y profundidad de la kinect, proporcionando una API que sirve de puente entre el hardware del equipo, NITE y las aplicaciones e interfaces del Sistema operativo.

También permite la captura de movimiento en tiempo real, el reconocimiento de gestos con las manos, el uso de comandos de voz y utiliza un analizador de escena que detecta y distingue las figuras en primer plano del fondo, dependiendo de la distancia a la que se encuentren de la kinect. Su proceso es establecer comunicación con NITE, enviar datos a NITE y recibir los datos de respuesta de este.

NITE

Es un Middleware de visión por ordenador en 3D que proporciona una API que controla los movimientos tanto de las manos del usuario como su cuerpo por medio de la kinect utilizando la información que obtiene OpenNI de la cámara depth, la de color, la cámara IR y la información de audio. Analiza la escena creando un esqueleto en el usuario y captando sus gestos para luego poder utilizarla en Processing, pudiendo hacer funciones como localización y seguimiento de la mano, distinguir a usuarios del fondo, etc.

La última versión es NITE 2.2.0.11, del día 11 de diciembre de 2012, y es la versión que utilizaremos para este trabajo. Se utiliza junto con OpenNI.

GITHUB

Es un sitio de almacenamiento web en el que puedes crear proyectos en un repositorio donde irás almacenando día a día los archivos con los que vas trabajando, guardando este las versiones de los archivos donde se destacan los cambios realizados entre estas. También te avisa de que archivos han sido creados y cuáles han sido destruidos. Todos estos cambios se guardan junto a la persona que los ha realizado en un historial.

En concreto he estado trabajando con la aplicación que ofrece GitHub para Windows, con la que anexionas una carpeta de tu ordenador con tu cuenta de la página web y así cada vez que modificas, creas o eliminas alguno de los archivos ocurre lo mismo en la web cuando sincronizas el ordenador con el repositorio. Esta aplicación permite también que varias personas trabajen y tengan un fácil acceso a un mismo archivo cada uno desde un ordenador distinto, en concreto ha ayudado a la hora de comunicarnos Javier y yo y a la hora de compartir los archivos con los que se ha trabajado, evitando así el tener que quedar para llevar un pendrive y pasarle los archivos de esta manera.

A la hora de trabajar yo trabajaba en mi ordenador e iba guardando todo lo que hacía en una carpeta creada por GitHub en mi ordenador en C:\Users\Usuario\ Documentos\GitHub\dart\picazas y cada vez que finalizaba algo, abría la aplicación y actualizaba el repositorio guardando los nuevos archivos y las nuevas versiones.

El único error obtenido con GitHub es que a partir del 08/03/2014 cada vez que intentaba actualizar el repositorio no me lo permitía, lo cual solucionamos creando uno nuevo llamado “Processing” dentro de mi cuenta. Los enlaces a los repositorios utilizados son:

- https://github.com/ingestado/dart/tree/master/picazas
- https://github.com/Picazas/Processing

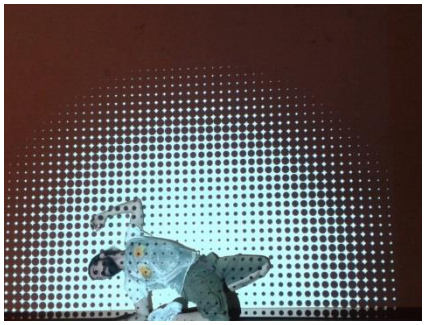
REUNIONES Y ENSAYOS CON EL LABORATORIO DE DANZA

La primera reunión tuvo lugar en los despachos del aula de las artes en la Universidad Carlos III de Madrid, el día 27/11/2013. En esta reunión el aula de las artes nos presentó la idea de unir tecnología y arte por medio de la kinect con la obra "Giselle", una obra de ballet clásico adaptada a la actualidad. Se nos pidió hacer hincapié en el tema de la muerte, ya que hay un momento de la obra en la que la protagonista muere y continúa la obra como un espectro.

La segunda reunión que tuvimos fue en un aula de danza en el mes de diciembre, en el cual colocamos el ordenador, el proyector y la kinect para enseñarles los avances que habíamos hecho. Principalmente se había creado escenarios que dibujaban la estela de la persona y alguno más de dibujar sobre el bailarín yayas, punteados, etc. Después de la demostración debatimos y apuntamos el camino a seguir a partir de entonces, creando nuevos escenarios con las ideas que nos daban y mejorando los que teníamos aplicando nuevos colores, metiendo y quitando formas, etc.

La tercera reunión se hizo en la misma aula de danza que en la segunda reunión, el día 17/03/2014 con el ordenador, el proyector y la kinect para enseñarles los avances que habíamos hecho, los cuales eran muchos tanto por la cantidad de nuevos escenarios que se poseía como por la mejora de los ya enseñados. En esta reunión ya se les presentó escenarios que no solo utilizaban la profundidad de la imagen y la máscara de usuario, sino que además se utilizaban imágenes, el centro de masas de las personas, fuerzas, etc. En dicha reunión tuvimos el problema de que el ordenador de trabajo y el de exposición son diferentes y este último hizo que algunos de los trabajos que funcionaban a la hora de trabajar ahora no funcionasen, lo cual dificulto la muestra de algunos escenarios. Al finalizar esta reunión nos pusimos como objetivo avanzar en algunos temas, mejorar los escenarios que más les había gustado y finalmente hacer que la imagen recogida por la kinect y la proyectada por el proyector coincidan a la perfección, evitando así que lo proyectado sea más pequeño que la persona.

Después de estas reuniones las siguientes fueron relacionadas no tanto con hacer una serie de "mundos", escenarios, sino con que íbamos a realizar de cara al público. La primera con este carácter fue el día 09/05/2014 en el centro cultural de Villa de Vallecas llamado "Sitio de mi recreo" de cara al festival "Óxido fest" que se realizará los días 20, 21 y 22 de junio de este año y en el cual participaremos. La participación en dicho festival aparece en el apartado "Puesta en escena" de este mismo trabajo.

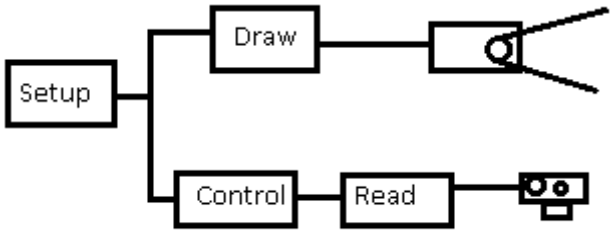


En esta reunión se nos enseñó el lugar del que disponíamos como un auditorio, una sala de gimnasio con espejos, la fachada del edificio, etc. Y nos explicaron como querían organizar el evento y sus horarios. Esta reunión nos sirvió para hacernos una idea de donde estar y de qué hacer, esta idea inicial fue por la noche hacer un espectáculo de cara al público en la fachada de 10 minutos y por la tarde en la sala de gimnasio dejarlo para la gente que quiera experimentar con los escenarios.

IMPLEMENTACIÓN GENERAL DEL SISTEMA

El funcionamiento del sistema en general es el siguiente. Nuestro sistema recibe la información del entorno en el que se está trabajando por medio de nuestro sensor, la cámara kinect. Una vez lo ha leído procesa la información y la traduce del lenguaje de la cámara al lenguaje de Processing, por medio de OpenNI, NiTE, SimpleOpenNI, etc.

Todo este control de la información recibida se inicializa por medio de la función "void setup ()", en la cual se crea en escenario y se llama a las diferentes librerías, entre ellas la kinect. Luego se lleva la información a la función "void draw ()" donde se trabaja con ella para proyectar más tarde por el proyector las formas y colores que queremos a partir de la imagen de la kinect o del movimiento que se genere en el entorno.



ELEMENTOS ESTÉTICOS

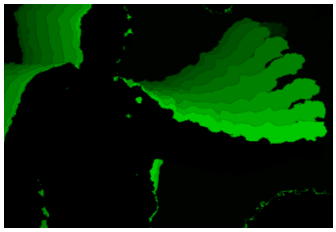
En este apartado explicaremos los elementos utilizados en los diferentes escenarios que se han creado en el trabajo, explicando cómo son visualmente y que opciones tienen con el movimiento del usuario. Más tarde, en el siguiente apartado, se explicará cómo funciona a nivel de código.

LA ESTELA

La estela es un efecto que utiliza el movimiento del cuerpo del usuario y va dejando un rastro por donde se mueve. El tiempo que esté presente dicha estela se puede regular. La estela se puede generar con la cámara User, cogiendo solo a los usuarios, o con la cámara Depth, con la que se cogería el movimiento de todos los cuerpos que se encuentren en el escenario. El color de la estela también se puede variar poniendo a todos los usuarios la misma, de diferentes colores, de varios colores, etc. Los escenarios creados son:

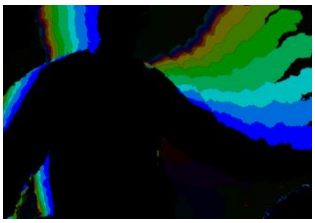
- "Estela todos del mismo color": Todos los usuarios aparecen con la misma estela del mismo color y además aparecen iluminados con el color del que tengan la estela. Es el más sencillo de todos ya que no hace distinción de personas.

- "Estela de un color si la forma de la persona": Todos los usuarios aparecen con la misma estela del mismo, pero sin iluminar a la persona que posee la estela. Es el más sencillo de todos ya que no hace distinción de personas.



- "Estela todos distinto color": Todos los usuarios aparecen con diferente estela cambiando el color dependiendo de qué usuario sea el que se encuentra en el escenario. Puede haber varias personas y cada una tener un color.

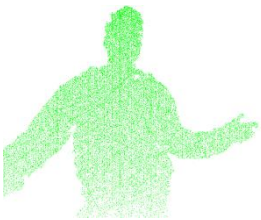
- "Estela multicolor": Todos los usuarios aparecen con la misma estela que va cambiando el color en lo que se mueve, por lo que va dejando una estela de muchos colores. La persona que posee la estela no está iluminada, así solo hay color a su paso, no sobre él.



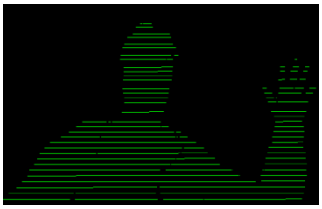
PROYECCIONES SOBRE EL USUARIO

Otros efectos se pueden crear proyectando únicamente sobre el usuario y no sobre el fondo del escenario. Este es el caso de estos tres escenarios:

- Persona punteada: Se crea una máscara sobre el usuario formada por puntos verdes (el color se puede variar) sobre el fondo que se haya dibujado. La densidad de puntos va decreciendo a medida que bajamos en el usuario, teniendo más puntos en la cabeza y menos y más separados en la cintura.



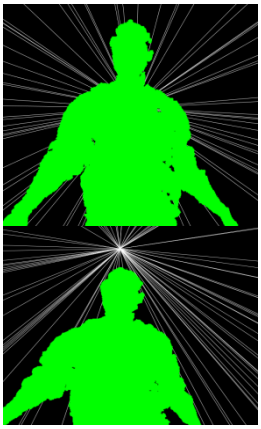
- Persona rayada: Se crea una máscara sobre el usuario formada por líneas oblicuas y blancas en movimiento sobre el fondo que se haya dibujado. El grosor, la distancia entre líneas y su color se puede variar. Otro cambio respecto al ejemplo anterior es que se le ha añadido un fondo a la máscara del usuario, en este caso negro. Otro escenario sería poniendo el fondo de la máscara persona y el color del fondo igual y creando líneas horizontales más gruesas. Este escenario se puede ver en la imagen de la izquierda.



RAYOS

Dentro de la temática de los rayos se pueden diferenciar dos tipos de rayos. Unos son creados con líneas rectas y los otros son creados por líneas curvas. Dichos rayos se pueden utilizar para multitud de escenarios y de aplicaciones, desde rayos de tormenta hasta rayos de luz o brillos. Ambos casos con sus escenarios son:

- Rayos de líneas rectas: Son rayos que se generan con una línea recta que va de un punto a otro. Dentro de este tipo hay varios escenarios. El primero son unos rayos blancos sobre un fondo negro que salen de la espalda del usuario a modo de representación de la luz que uno lleva dentro.



Una variante que se añadió después fue que en caso de que el usuario alzase las manos la luz se empieza a elevar hasta que desaparece. El usuario se encuentra iluminado con un color verde para diferenciarlo bien, pero este color se puede variar.

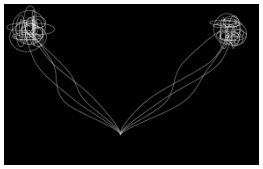
Otro escenario con este tipo de rayos es el llamado "Brillo como el agua", ya que utilizando rayos negros sobre un fondo verde se genera un efecto parecido al del brillo que se genera en el agua con la luz. Una variante en este escenario es que el foco de estos brillos sigue al usuario desde la parte superior de la imagen. Al usuario se le ilumina con luz blanca con el objetivo de que se le vea en el escenario, aunque esto se puede cambiar o quitar.

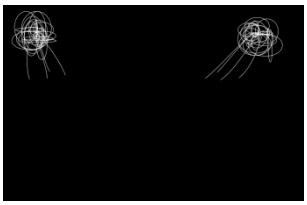
- Rayos curvos: Son rayos creados con formas curvas, son más fluidos que los anteriores, mostrando así menos fuerza y más movimiento. Van cayendo rayos uno a uno hasta y cuando llega cada rayo al suelo se produce un fogonazo, se pone la pantalla en blanco. Ilumina al usuario para que se le pueda ver a la perfección en el escenario, ya que si no sería difícil por el fondo negro. En la imagen el rayo se encuentra a la derecha de esta. Este escenario se puede mezclar con algún fondo tétrico como de algún bosque de invierno para darle más sentimiento.

HILOS

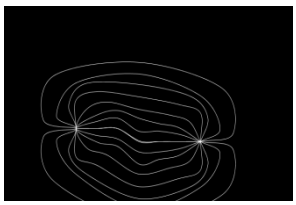
En relación con los hilos se ha creado varios escenarios. El objetivo principal de estos escenarios es que el laboratorio de danza pueda trabajar con el mito de la hilandera con la ayuda de las proyecciones y las uniones que se pueden crear con estos. Los escenarios creados son:

- Ovillos de hilo: En este escenario se generan ovillos de hilo en movimiento en la parte superior y cuando algún usuario entra en el escenario estos atrapan al usuario con un hilo cada uno, estando estos en continuo movimiento también. En este escenario no se ilumina al usuario ya que está pensado para un escenario donde se le vea a este. En la imagen de la izquierda se puede observar cómo van los hilos hacia la situación del usuario y en la imagen de la derecha se observa como quedan los hilos una vez unidos.

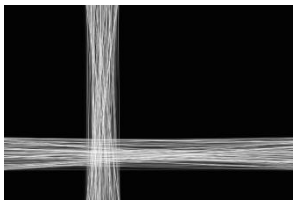


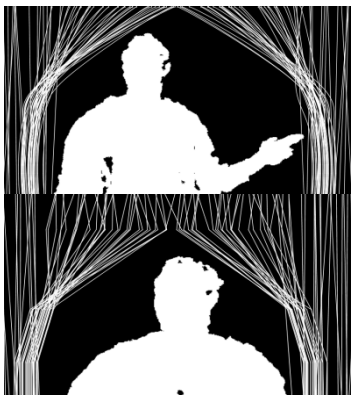


- Hilos que unen usuarios: Este escenario capta a dos usuarios para luego unireles por medio de "hilos" que se encuentran en constante moviemiento. Se utiliza el centro de masas como punto, por lo que el origen de los hilos de cada usuario le seguirá.



- Líneas de varios tipos: En este escenario hay tres escenarios que se van variando cuando pulsas la tecla "a" o "s". El primero coge la situación del usuario o usuarios y para cada uno genera líneas horizontales y verticales aleatorias en movimiento teniendo como corte el centro de masas del usuario. Son líneas blancas sobre fondo negro.





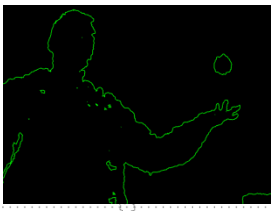
El segundo escenario crea una cortina de hilos blancos sobre un fondo negro. Estos hilos se abren en el espacio que ocupa el usuario, esquivando a este. Utiliza la sección eficaz del usuario o usuarios que se encuentren en el escenario. El tercer escenario es igual que el segundo pero los hilos al abrirse en vez de ser más suaves, con curvas, son más bruscos, con líneas rectas. La imagen superior es el segundo escenario y la inferior es el tercer escenario, aquí se puede ver la diferencia donde se abren los hilos.

PINTANDO DISTANCIAS

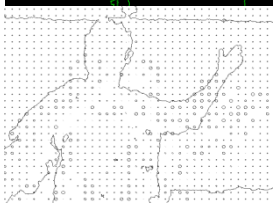
En este escenario se capta la distancia con la kinect y se va dibujando todo el escenario de varios colores. Estos colores dependen de la distancia a la que se encuentren, siendo el más cercano el granate y el más alejado el amarillo y blanco. Se dividen en menos de un metro, de uno a dos metros (naranjas) y más de dos metros amarillos.

ESCENARIOS CON SILUETA

En estos escenarios se genera una silueta del usuario y de los objetos de alrededor mostrándola en su mayoría de blanco sobre el fondo negro. Dependiendo del escenario, esta silueta se utilizará para diferentes fines. Los escenarios son:



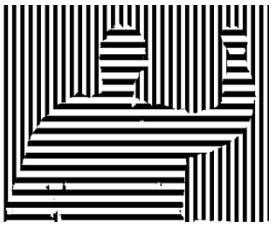
- Silueta cámara depth: Crea la silueta de todos los objetos de la imagen a una distancia menor a 2.5 metros, esta distancia se puede cambiar al igual que los colores de la silueta y del fondo.



- Círculos en movimiento con la silueta: Crea una silueta de todos los cuerpos del escenario sobre un fondo blanco con círculos negros. Cuando pasa la silueta por los círculos, estos empiezan a crecer y a decrecer durante un periodo de tiempo.

- Nieve: Se generan partículas de nieve continuamente que van cayendo poco a poco y también se genera la silueta del usuario. Cuando la nieve entra en contacto con la silueta esta desaparece dejando una luz que va creciendo hasta que desaparece esta también

UNIÓN CUERPO-FONDO

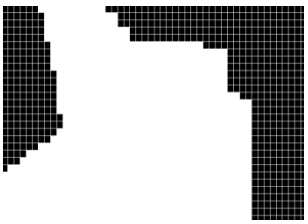


En estos escenarios el objetivo es crear contrastes entre la proyección que se crea sobre el cuerpo y el fondo que se está creando, teniendo cada uno unas características diferentes al otro. El escenario creado con este objetivo es el que se observa en la foto de la derecha. Se generan líneas verticales para el fondo y líneas horizontales para el usuario, creando ese choque de líneas.

INTERACCIÓN CON LA SECCIÓN DEL CUERPO

En estos escenarios el objetivo es utilizando la información que nos da la máscara del usuario crear cambios visuales en el escenario. Los escenarios son:

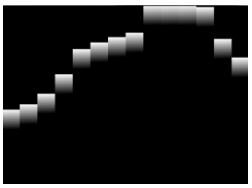
- Tablero de luz: Se divide la imagen en X cuadrados iguales en una matriz de "A"x"B". Estos cuadrados se iluminarán cuando el usuario posea alguna parte de su cuerpo sobre estos apareciendo así su forma más o menos pixelada, dependiendo del número de cuadrados que se hayan creado.

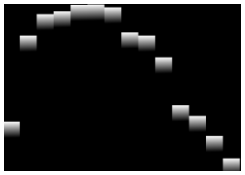


- Luz que se desvanece: Este escenario está pensado para representar como la luz del usuario se va desvaneciendo, va empequeñeciendo. En el escenario se dibuja una máscara del usuario y de los objetos de negro sobre un fondo blanco, como se puede ver en la imagen de la derecha. Una vez el usuario se agache y su superficie sea menor de "x" el fondo blanco irá disminuyendo su superficie con forma de círculo hasta desaparecer, como se puede observar en la imagen de la izquierda.

- Luz que se desvanece v2.0: En esta nueva versión que se ha creado tiene como objetivo tener mayor interacción con el usuario que la anterior. A las características que poseía la versión anterior se le ha añadido un par más. La primera es que el círculo al cerrarse persigue a la persona con su centro de masas, como se puede observar en la imagen, y la segunda es que se le ha aumentado la velocidad a la que se cierra, para que se puedan generar varios seguidos de forma rápida.

- Cubos que caen: En este escenario se han creado unos cubos de tonos grises que se van difuminando, siendo más luminosos en la parte superior y negros en la inferior. El escenario está pensado para que en la zona en la que se encuentre el usuario los cubos asciendan y en aquellas zonas en las que el usuario no está vayan descendiendo. En la foto de la derecha el usuario estaría a la derecha de la imagen y en la otra estaría en el lado opuesto. El usuario no sale iluminado y por eso en la imagen no se ve la silueta.





- Cubos que caen v2.0: En esta nueva versión que se ha creado tiene como objetivo tener mayor interacción con el usuario que la anterior. A las características que poseía la versión anterior se le ha añadido una gravedad que hace que la velocidad de caída vaya aumentando a medida que va cayendo el cubo y un regulador de la velocidad de subida por medio del flujo óptico, cuanto más movimiento haya en el escenario mayor será la velocidad de subida de los cubos.

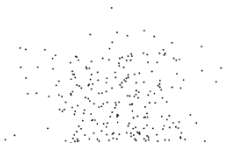
SECCIÓN EFICAZ

La sección eficaz es el área que abarca todos los pixeles del cuerpo del usuario o los cuerpos del usuario. Los escenarios que funcionan con la sección eficaz son:

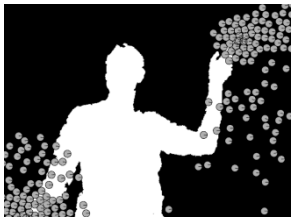
- Cuadrados cercando personas: En este escenario se cerca la sección eficaz de los usuarios con un cuadrado, todos los usuarios poseen un cuadrado común y este varía dependiendo del espacio que ocupen. En la imagen se ve dibujada una máscara de las persona, aunque también se utiliza sin iluminar, solo con los cuadrados. También se utiliza con cuadrados independientes como se puede ver en la imagen de la izquierda.

- Luces laterales: En este escenario sale luz de los laterales del escenario que se van difuminando hasta llegar al usuario, donde habrá poca o ninguna luz dependiendo como este. Dependiendo de si está con los brazos estirados o pegados al cuerpo habrá más o menos luz en su cuerpo, ya que utiliza la sección eficaz. De este escenario salieron 3 variantes más, con el difuminado mayor, con más luz encima del usuario y con menos. En las imágenes se ha iluminado al usuario de rojo para que se le vea mejor, en el escenario no poseería ningún color.

- Fuerzas en partículas: El escenario estará lleno de partículas en movimiento. Si el usuario está en una postura normal, las bolas se verán atraídas por este y se moverán hasta su situación flotando donde él se encuentre, en su centro de masas. En el momento que el usuario abra los brazos estirándolos, las partículas saldrán despedidas en todas direcciones con una fuerza proporcional al movimiento del usuario, por el grado de amplitud de sus brazos y por la velocidad del movimiento. La imagen de la izquierda es con el usuario quieto y la de la derecha es la imagen cuando abre los brazos.



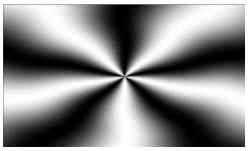
- Creador de cuerpos: El usuario va creando cuerpos en los extremos más alejados del centro de masas, utilizando para ello la sección eficaz. Los cuerpos utilizados son circunferencias con una línea para poder ver la rotación de estos en lo que caen debido a una gravedad que se ha creado.



MOVIMIENTO USUARIOS

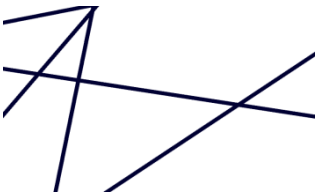
En estos escenarios se utiliza el movimiento que hay en escena y la cantidad de usuarios que se encuentran en esta. Los creados con estas características son:

- Fondo variante: En este escenario se genera un fondo que se va deformando a medida que entran individuos al escenario, estando en perfecto estado cuando no hay nadie y deformándose mucho cuando hay varios. En la imagen de la derecha aparece el fondo creado normal y en la foto de la izquierda aparece con varios usuarios. El fondo se puede variar por otros.



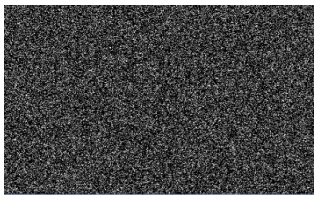
- Figuras en movimiento: Se crean unas formas rectangulares que están en continuo movimiento moviéndose de un lado a otro de la pantalla. Esta velocidad varía dependiendo del movimiento que haya en la imagen captada por la kinect, moviéndose las figuras más rápido si los usuarios se mueven rápido y muy despacio si los usuarios no se mueven. En este ejemplo se puede ver al usuario y objetos en verde para que sea más visual el movimiento de este, en el escenario todo ello no se vería.

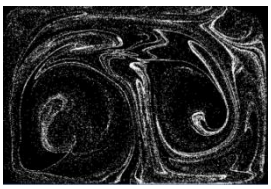
- Fogonazos con movimiento: El programa va calculando el movimiento que hay en las imágenes y cuando este movimiento supera una cantidad establecida se produce un fogonazo con una imagen de fondo blanco con líneas negras aleatorias que va desapareciendo poco a poco hasta que se vuelve a quedar todo negro. En la imagen se puede ver como es la imagen del fogonazo creado.



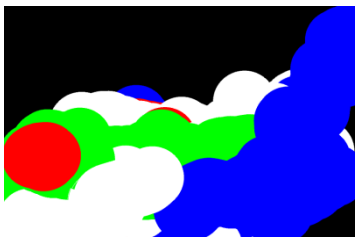
También se ha creado un escenario que utiliza el movimiento que realiza el usuario y la dirección que toma dicho movimiento, flujo óptico. Este escenario es:

- Partículas flotando: Se generan una gran cantidad de partículas (65.536) que se encuentran en el escenario, imagen de la derecha. Cuando el usuario aparece el programa reconoce el cuello, las manos y los pies y sigue el movimiento que estos efectúan moviendo todas las partículas que se encuentran a su paso, imagen de la izquierda. Cuando alguna partícula ha sido movida sigue su movimiento con inercia, no se para en seco finalizando su movimiento.





PROYECCIONES DESDE EL TECHO

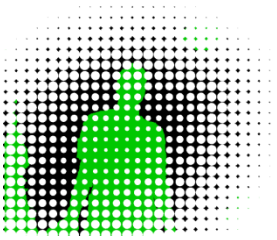


Otra manera de proyectar sobre el escenario es por medio de un proyector y una cámara que se encuentren anclados en el techo y apunten hacia el suelo, captando el movimiento de los usuarios desde arriba. El escenario creado con este objetivo es "círculo de colores", en el cual se diferencian a los distintos usuarios por colores y se dibuja una circunferencia de dicho color donde estos se hallan. Al moverse esta circunferencia les sigue y va pintando el suelo que pisa de su color, pudiendo hacer una combinación de colores al moverse. Este seguimiento se lleva a cabo con el cálculo del centro de masas.

POSICIÓN USUARIO

Se pueden unir el cuerpo y el fondo del modo que, dependiendo donde se sitúe el cuerpo, el fondo irá variando. Este es el caso de los escenarios que se puede observar a continuación:

- Fondo de círculos blancos: En este escenario se hacen más pequeños todos los círculos que se encuentran cerca del centro de masas del usuario, abriendo así el manto blanco que lo dibuja todo para poder ver al usuario. En la puesta en escena no se ilumina a la persona de verde como se ve en la imagen, eso solo se ha hecho para que en la foto se pueda ver la respuesta del escenario con el usuario.



- Partículas que persiguen: En este escenario se capta el centro de masas del usuario para a partir de este punto crear un área en la que estarán volando partículas continuamente alrededor de este. Si el usuario se mueve estas le perseguirán, ya que al cambiar el centro de masas de posición el área variará también.

PROYECCIÓN DE IMÁGENES

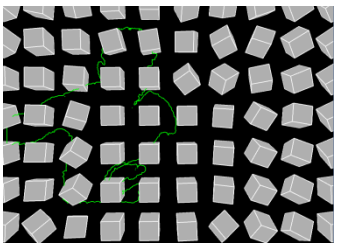
Processing también permite la proyección de imágenes y vídeos, este es el caso del siguiente escenario, que utiliza imágenes para expresar cosas que con formas sería más difícil. El escenario se llama "caras", ya que las imágenes utilizadas son máscaras. En este escenario van apareciendo imágenes de caras, de tamaños aleatorios y con velocidades aleatorias, de un lado a otro de la imagen cruzándose entre ellas. En su movimiento van dejando una pequeña estela que les da un enfoque más fantasmagórico.

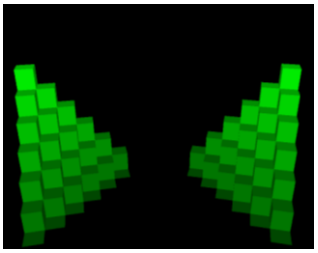
Otro escenario utilizado con estas máscaras es variando el movimiento. En vez de moverse de un lado al otro de la imagen, avanzan a medida que se acercan desde el fondo de la imagen hasta "chocar" con el escenario incrementando su tamaño por el camino.

3D

En el área del 3D se han creado un par de escenarios utilizando la librería OPENGL que permite crear objetos en 3D y moverlos con total libertad. Los dos escenarios creados son:

- Cubos en movimiento al pasar: Se crean cubos en 3D en todo el escenario y se dibuja la silueta del usuario. Cuando el usuario se sitúa encima de algún cubo, este empieza a rotar en todos los ejes con una velocidad constante. Si el usuario deja de estar en el área del cubo este se parará y se quedará con la rotación que tenía en el último momento.





- Pirámides variantes: Se crean dos pirámides como se muestra en la imagen de la izquierda que varían su tamaño dependiendo de dónde se encuentre el usuario (se utiliza el centro de masas).

MEZCLA DE ESCENARIOS

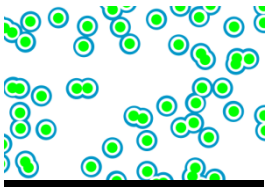
Estos escenarios tienen como objetivo unir el resto de escenarios en un único programa que va variando dependiendo de variables o de que teclas se pulsan mientras que el programa está funcionando. Uno de los escenarios creados para este fin es el escenario "centro cultural", el cual tiene 10 escenarios, 5 para una persona y 5 para más de una persona. El control se lleva a cabo por un contador que hace de temporizador y por una función que dependiendo de la cantidad de usuarios que se hallen en el escenario cambia o no.

Este escenario ha sido creado para proyectarlo en un centro cultural y que la gente que pase por allí pueda experimentar con él.

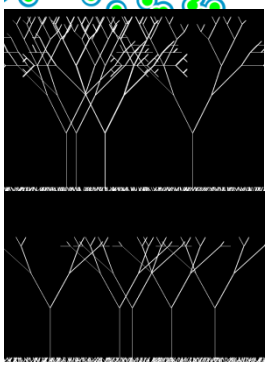
FONDOS DE ESCENARIO

Además de crear escenarios que interactúen con el usuario también se han creado escenarios que tienen la única función de decorar el escenario. Estos escenarios se han creado sobre todo para la unión de la parte decorativa con otro escenario en la que sí interactúe el usuario, por ejemplo, un fondo de árboles con el escenario de la estela pintándola de color blanco. Los escenarios son:

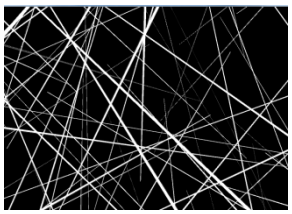
- Células cayendo: Se generan continuamente células de color verde blanco y azul que caen continuamente con un movimiento oblicuo. No dejan de caer, ya que se están entrando y saliendo continuamente.



- Árboles y césped: Se crean cinco árboles en sitios aleatorios que van creciendo a la vez que el césped hasta quedar con la forma y el tamaño que se puede observar en la imagen. Este escenario surgió a raíz de que el segundo acto de la obra "Giselle" tuviese un escenario de un bosque por la noche. La primera imagen es como terminan las plantas y la segunda imagen es en lo que van creciendo.



- Líneas 3D: Se generan líneas con tamaño, grosor, posición y orientación aleatorias formando una especie de tela de araña. Tanto en número de líneas como el color se puede variar. De este fondo se crean varias variantes. Una es con las líneas fijas, otra es creando u destruyendo líneas, así varía el escenario y la última es el cambio de todas por medio de fogonazos, cambios rápidos.



- Palabras: Se van generando palabras de forma aleatoria que van avanzando en el eje z dejando una estela a su paso. Se puede crear un orden de salida de las palabras para querer mandar un mensaje o se pueden generar las palabras con orden aleatorio.

- Ondas: Se crean ondas de agua en una posición y con unos tamaños aleatorios en el escenario. Este escenario transmite caos y desorden al no saber cuándo saldrán las ondas, dónde, con qué tamaño, etc.

CONTROL DE LOS ELEMENTOS ESTÉTICOS

En el apartado anterior se han explicado los elementos utilizados en los diferentes escenarios que se han creado en el trabajo, explicando cómo son visualmente y que opciones tienen con el movimiento del usuario. Ahora en este apartado se va a explicar cómo se ha conseguido por medio del código ese resultado en los escenarios, haciendo una explicación general sin entrar en todos los escenarios que se han creado. Las principales herramientas y cálculos han sido las que se muestran a continuación.

CALIBRACIÓN PANTALLA

La imagen que se captura con la kinect tiene una resolución de 640x480, por lo que sólo se podría proyectar en pantallas de 640x480. Para solucionar este problema y poder adaptar esta imagen a cualquier tamaño se aplica a dicha imagen una escala que se calcula en el programa. Para que este cálculo se lleve a cabo se utilizan tres variables:

- Int kinectWidth: Representa el ancho de la imagen de la kinect, 640.
- Int kinectHeight: Representa el alto de la imagen de la kinect, 480.
- Float reScale: Es el valor de la escala que se va a aplicar a la imagen.

Para realizar estos cálculos se llevan a cabo varios pasos:

1. Se declaran las variables como variables globales.

```
int kinectWidth = 640;
int kinectHeight = 480;
// to center and rescale from 640x480 to higher custom resolutions
float reScale;
```

2. Se calcula el valor de reScale con el ancho de la imagen que se desea y con el ancho de la imagen de la kinect.

```
reScale = (float) width / kinectWidth;
```

3. Se mueve la imagen en el eje y con los siguientes cálculos y se aplica la escala sobre esta.

```
translate(0, (height-kinectHeight*reScale)/2);
scale(reScale);
image(cam,0,0);
```

MODIFICADOR DEL ÁNGULO DE LA KINECT

La función de este modificador es poder variar el ángulo de inclinación de la cámara kinect. Este ángulo puede estar entre 0 y 30 grados y utiliza un motor para cambiarlo, por ello hay que hacer un programa que lo mueva y no se puede mover a mano ya que se podría estropear dicho motor.

Para este programa se utiliza la librería "openkinect", ya que posee la función "tilt()" que es la que mueve el motor de la kinect. Los pasos para llevar a cabo la función de mover el ángulo de la kinect son:

1. Crear una variable tipo float que guarde el ángulo de inclinación de la cámara.

```
float angle = 0;
```

2. Iniciar en "setup()" la cámara kinect habilitando una de las cámaras que posee para poder ver donde enfoca la kinect y poder así el mejor grado de inclinación.

```
kinect = new Kinect(this);
kinect.start();
kinect.enableDepth(true);
```

3. Crear una función que utilice las teclas del teclado para aumentar o disminuir el ángulo con la función "tilt()".

```
void keyPressed(){
switch(key)
  {
    case'a':
      angle++;
      kinect.tilt(angle);
      break;
    case's':
      angle--;
      kinect.tilt(angle);
      break;
  }
}
```

MÁSCARA DEL USUARIO

Se genera por medio de la cámara User. El objetivo de esta herramienta es crear una máscara que tenga un color y que represente la forma del usuario o de los usuarios que se encuentran en la zona, ya sea para generar líneas en estos con dicha información o para utilizarla en el entorno. Las variables globales para este escenario son:

- Las variables de la calibración.
- Int [] userMap: Es un array que almacena los datos de todos los pixeles de la imagen de la cámara. Guarda un "1" si el pixel pertenece a un usuario y un "0" si no pertenece a un usuario.
- Int [] userList: Guarda un listado de los usuarios que se encuentran en el escenario.
- PImage cam: Se crea una imagen sobre la que se realizarán todos los cambios del programa para luego proyectarla.

En "void setup ()" se activa la cámara depth y se calibra la pantalla al tamaño deseado. El modo de generar la estela es por medio de los siguientes pasos:

1. Se guarda en userMap los valores de la máscara de usuario, para ir dibujando con esta la estela y en userList el listado de usuarios que se encuentran en el escenario.

```
userMap = kinect.userMap();
userList = kinect.getUsers();
```

2. Se va pasando de un usuario a otro para hacer el barrido después y poder diferenciarles por medio de un "for".

```
if (kinect.getNumberOfUsers() > 0) {
  cam.loadPixels();
  for(int z = 0; z < userList.length; z++) {
```

3. Se hace un barrido de la imagen y todos aquellos pixeles de userMap que posean el valor "1" se pintan del color deseado, estos pixeles serán la estela que se dibuja.

```
for (int i = 0; i < userMap.length; i++) {
  if (userMap[i] != 0) {
    // make it green
    cam.pixels[i] = color(0, 255, 0); //The color you want for all pleople
  }
}
```

Una vez ya se tiene esta imagen se puede proyectar como usuarios de colores, con proyecciones, etc. este es el modo de conseguir dicha información, ahora se puede variar como utilizarla.

ESTELA

A la hora de crear la estela de los usuarios se puede realizar de dos maneras diferentes que varían una de la otra por la cámara que utilizan y por los objetos que captan y utilizan cada una.

Uno de los modos es por medio de la cámara "Depth", la cual una vez activada capturará todas las distancias de los pixeles de la imagen. Las variables globales para este escenario son:

- Las variables de la calibración.
- Int maxValue: para la cámara depth, es el valor de la distancia elegida para la imagen, todos aquellos pixeles que estén a menos distancia que "maxValue" se verán.
- Int [] depthValues e int[] depthValues2: Son dos arrays que almacenan la información de la cámara depth para con esta ir generando la estela.
- PImage cam: Se crea una imagen sobre la que se realizarán todos los cambios del programa para luego proyectarla.

En "void setup ()" se activa la cámara depth, se calibra la pantalla al tamaño deseado y se le da un valor a maxValue de 2500 (2.5 metros). El modo de generar la estela es por medio de los siguientes pasos:

1. Se genera un rectángulo del color del fondo con una opacidad de 35 que abarque toda la pantalla, así la estela que se va dibujando se irá desapareciendo poco a poco.

```
fill(0,0,0,35);
rect(0,0,width,height);
```

2. Se guarda en depthValues2 los valores de depthValues, para ir dibujando con esta la estela y no tener que dibujar así sobre la persona ningún color.

3. Se guardan los nuevos valores de la cámara depth en depthValues.

```
depthValues = kinect.depthMap();  //Save the Depth's values in an array
```

4. Se hace un barrido de la imagen y todos aquellos pixeles de depthValues2 que estén a una distancia menor de maxValue se pintan del color deseado, estos pixeles serán la estela que se dibuja.

```
for(int x = 0; x < 640; x++){                       //See all the pixels
  for(int y = 0; y < 480; y++){
    clickPosition = x + (y*640);                    //We see which pixel we are working on
    clickedDepth = depthValues2[clickPosition];     //See the pixel's value
    if (clickedDepth > 455){
    if (maxValue > clickedDepth){
          cam.pixels[ clickPosition] = color(255, 0, 0);
    }
   }
  }
 }
```

5. Se realiza el mismo barrido del paso 3 pero con los valores de depthValues y se pintan de negro, así la silueta de la persona no saldrá de colores.

6. Se proyecta la imagen que se ha ido modificando durante todo el proceso (cam).

```
cam.updatePixels();
translate(0, (height-kinectHeight*reScale)/2);
scale(reScale);
image(cam,0,0);
```

El otro modo es por medio de la cámara User. El modo es muy parecido, la diferencia es que en vez de utilizar la profundidad de los pixeles se utilizan los pixeles de la máscara del usuario. Las variables globales para este escenario son:

- Las variables de calibración de pantalla.
- Int [] userMap: Es un array que almacena los datos de todos los pixeles de la imagen de la cámara. Guarda un "1" si el pixel pertenece a un usuario y un "0" si no pertenece a un usuario.
- Int [] userList: Guarda un listado de los usuarios que se encuentran en el escenario.
- PImage cam: Se crea una imagen sobre la que se realizarán todos los cambios del programa para luego proyectarla.

En "void setup ()" se activa la cámara depth y se calibra la pantalla al tamaño deseado. El modo de generar la estela es por medio de los siguientes pasos:

1. Se genera un rectángulo del color del fondo con una opacidad de 35 que abarque toda la pantalla, así la estela que se va dibujando se irá desapareciendo poco a poco.

```
fill(0,0,0,35);
rect(0,0,width,height);
```

2. Se guarda en userMap los valores de la máscara de usuario, para ir dibujando con esta la estela y en userList el listado de usuarios que se encuentran en el escenario.

```
userMap = kinect.userMap();
userList = kinect.getUsers();
```

3. Se va pasando de un usuario a otro para hacer el barrido después y poder diferenciarles por medio de un "for".

```
if (kinect.getNumberOfUsers() > 0) {
  cam.loadPixels();
  for(int z = 0; z < userList.length; z++) {
```

4. Se hace un barrido de la imagen y todos aquellos pixeles de userMap que posean el valor "1" se pintan del color deseado, estos pixeles serán la estela que se dibuja.

```
for (int i = 0; i < userMap.length; i++) {
  if (userMap[i] != 0) {
    // make it green
    cam.pixels[i] = color(0, 255, 0); //The color you want for all pleople
  }
}
```

5. Se proyecta la imagen que se ha ido modificando durante todo el proceso (cam).

```
cam.updatePixels();
translate(0, (height-kinectHeight*reScale)/2);
scale(reScale);
image(cam,0,0);
```

CENTRO DE MASAS

El centro de masas se consigue por medio de un vector que guarda sus ejes {x,y,z}, para cada usuario que este en el escenario. En caso de que solo se tenga que trabajar con un centro de masas, este se podrá guardar en un vector, pero si son varios centros de masas los que se necesitas, se guardarán todos ellos en un array. Para calcular el centro de masas, ya sea de un modo u otro, es siguiendo los siguientes pasos y con las siguientes variables. Las variables globales son:

- Las variables de calibración de pantalla.
- PVector position: Vector en el que se guarda la posición del centro de masas del usuario con los parámetros que nos da la kinect.
- PVector jointPos: Se utiliza si solo se va a trabajar con un punto. Es un vector en el que se guardan las coordenadas del centro de masas con la escala ya aplicada.
- Int [][] jointPos: Se utiliza si solo se va a trabajar con más de un punto, es una variable con dimensiones int[8][3], {x,y,z} para 8 usuarios. Son los puntos en los que se guardan las coordenadas de los centros de masas con la escala ya aplicada.
- Int [] userList: Guarda un listado de los usuarios que se encuentran en el escenario.

Para el cálculo se lleva a cabo el cálculo de reScale como ya se ha explicado en apartados anteriores y userList almacena el listado de los usuarios que se encuentran en el escenario. El modo de calcular el centro de masas para cada usuario es:

1. Usuario por usuario se consigue el centro de masas guardándolo en la variable position.

```
if(userList.size()>0){
for (int i=0; i<userList.size(); i++){
  int userId = userList.get(i); //getting user data
  kinect.getCoM(userId, position);
  kinect.convertRealWorldToProjective(position, position);
```

2. Si solo se necesita un centro de masas, se guarda position en jointPos aplicándole la escala.

```
jointPos.x = position.x*reScale;
jointPos.y = position.y*reScale;
```

3. Si se necesitan varios centros de masas, se guarda position en jointPos aplicándole la escala en el sitio que le corresponda en el array.

```
jointPos[i][0] = position.x*reScale;
jointPos[i][1] = position.y*reScale;
```

Una vez que se tienen los puntos se pueden utilizar para muchas cosas como punto de seguimiento de alguna forma para que persiga al usuario o como punto de unión con otros usuarios.

CÁLCULO DEL ESQUELETO

Como ya se ha explicado en apartados anteriores, la kinect tiene la función de poder dar la posición de algunos puntos del esqueleto del usuario. En el cálculo del esqueleto es en el que más problemas se han encontrado durante el trabajo, ya que de versiones anteriores a versiones más actuales posee muchos cambios. En este apartado se explicará cómo calcular el esqueleto de los usuarios con la versión 2.1. Las variables globales necesarias para el esqueleto son:

- Las variables de calibración de pantalla.
- PVector position: Vector en el que se guarda la posición del punto del esqueleto del usuario.
- PVector jointPosXX: Es un vector que se utiliza para guardar las coordenadas de algún punto del esqueleto, de ahí que acabe en XX. Por ejemplo, si vamos a calcular el punto de la mano derecha (Right Hand) en el código se llamará a este vector "jointPosRH".
- Int [] userList: Guarda un listado de los usuarios que se encuentran en el escenario. Al igual que en el centro de masas, cada usuario posee su esqueleto.

Se va a utilizar como ejemplo la búsqueda de las coordenadas del cuello, aunque se puede calcular de la cabeza, las manos, los codos, los hombros, el cuello, el pecho, la cadera, las rodillas y los pies. Para calcular el esqueleto del usuario se siguen los siguientes pasos:

1. Se detecta al usuario por medio de la función "onNewUser" para que el programa calcule los puntos del esqueleto del usuario.

```
void onNewUser(SimpleOpenNI curContext, int userId)
{
  println("onNewUser - userId: " + userId);
  println("\tstart tracking skeleton");

  curContext.startTrackingSkeleton(userId);
}
```

2. Para el usuario deseado o para cada uno de los usuarios se guarda en position las coordenadas del punto que se quiere guardar, en este caso las coordenadas del cuello.

```
for(int i=0;i<userList.length;i++)
  {
    int userId = userList [i];
   kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_NECK,position);
   println("Neck:"+position);
  }
```

3. Se guarda position en jointPosXX aplicándole la escala, en este caso jointPosN.

```
jointPosN.x = position.x * reScale;
jointPosN.y = position.y * reScale;
```

El esqueleto es muy poco preciso y se pierde con facilidad, por ello se ha utilizado muy poco en este trabajo, aparte de que una vez que ha aparecido el usuario se necesita un tiempo para que el programa cree el esqueleto sobre el usuario y no es instantáneo. Además se pueden utilizar las funciones "onLostUser" y "onVisibleUser", que te avisan cuando se pierde a algún usuario o cuando aparece algún usuario.

```
void onLostUser(SimpleOpenNI curContext, int userId)
{
  println("onLostUser - userId: " + userId);
}

void onVisibleUser(SimpleOpenNI curContext, int userId)
{
  println("onVisibleUser - userId: " + userId);
}
```

Estas funciones no son necesarias para el cálculo del esqueleto del usuario, aunque ayudan bastante a la hora de saber si ha cogido al usuario que ha entrado o no ya que el esqueleto tarda en calcularse.

SILUETA

La silueta es el cálculo del contorno de objetos y usuarios que se encuentren en una imagen, ya haya sido cogida con la cámara Depth o con la cámara User. Para el cálculo de la silueta se necesitan las siguientes variables:

- Las variables de calibración de pantalla.
- Int [][] shape: Es un array de dimensiones [Nº de pixeles de la pantalla][3] y sirve para guardar la coordenada "x", la coordenada "y" y si forma parte o no de la zona coloreada.
- PImage cam: Se crea una imagen sobre la que se realizarán todos los cambios del programa para luego proyectarla.
- PImage cam2: Se crea una imagen sobre la que se dibujará la silueta para luego proyectarla.

Los pasos para calcular la silueta en una imagen son:

1. Guardar en la imagen "cam" la captura de la cámara de la kinect, ya sea Depth o User, y colorear de un color los objetos y de otro el fondo de la captura.

```
for (int z=0; z<userList.length; z++){
  for(int h = 0; h < 480; h++){           //See all the pixels
     for(int w = 0; w < 640; w++){
       clickPosition = w + (h*640);        //We see which pixel we are working on
       if (userMap[clickPosition] != 0) {
           cam.pixels[clickPosition] = color(255);
         }
         else cam.pixels[clickPosition] = color(0);
       }
   }
```

2. Hacer un barrido de la imagen para calcular que pixeles se encuentran en el borde entre el objeto y el fondo y guardarlos en “shape”.

```
for(int h = 0; h < 480; h++){            //See all the pixels
 for(int w = 0; w < 640; w++){
   clickPosition = w + (h*640);         //We see which pixel we are working on
   clickedDepth = depthValues[clickPosition];    //See the pixel's value

     if (userMap[clickPosition] != 0){
       now = 1;
     }
     else now = 0;

   if(before < now){
     shape[clickPosition][0] = w-1;
     shape[clickPosition][1] = h;
     shape[clickPosition][2] = 1;
   }
   else if(before > now){
     shape[clickPosition][0] = w;
     shape[clickPosition][1] = h;
     shape[clickPosition][2] = 1;
   }
   before = now;
 }
} //For end
```

3. Colorear en “cam2” los puntos obtenidos para tener la silueta que se encuentran guardados en “shape”.

Aunque es un modo de calcular los puntos que componen la silueta, también se puede calcular la silueta por medio de la librería “BlobDetection”. Con esta librería los pasos a seguir serían:

1. Llamar a la librería BlobDetection.

```
import blobDetection.*;
BlobDetection theBlobDetection;
```

2. Asignar la imagen que se va a trabajar a la librería y poner a esta las características de la imagen.

```
theBlobDetection = new BlobDetection(cam.width, cam.height);
theBlobDetection.setPosDiscrimination(false);
theBlobDetection.setThreshold(0.38f);
theBlobDetection.computeBlobs(cam.pixels);
```

3. Guardar en la imagen “cam” la captura de la cámara de la kinect, ya sea Depth o User, y colorear de un color los objetos y de otro el fondo de la captura.

```
cam.loadPixels();
for (int z=0; z<userList.length; z++){
  for(int h = 0; h < 480; h++){            //See all the pixels
     for(int w = 0; w < 640; w++){
       clickPosition = w + (h*640);         //We see which pixel we are working on
       if (userMap[clickPosition] != 0) {
           cam.pixels[clickPosition] = color(255);
         }
         else cam.pixels[clickPosition] = color(0);
       }
    }

}
cam.updatePixels();
```

4. Utilizar la función computeBlobs para calcular los puntos que pertenecen a la silueta.

```
theBlobDetection.computeBlobs(cam.pixels);
drawBlobsAndEdges(true, true);
```

5. Hacer un barrido de los pixeles de theBlobDetection, que son los pixeles de la imagen, para ver cuales pertenecen a la silueta o no.

```
for (int n=0 ; n<theBlobDetection.getBlobNb() ; n++)
{
  b=theBlobDetection.getBlob(n);
  if (b!=null)
  {
```

Finalmente ambos modos llevan a una misma solución, siendo más cómodo el uso de la librería BlobDetection.

SECCIÓN EFICAZ

La sección eficaz es el área que ocupa un usuario, en este trabajo se ha utilizado con forma de cuadrado tanto individualmente para cada usuario como en conjunto, con un rectángulo que abarque a todos los usuarios. En el primer ejemplo que se va a utilizar se explicará el cálculo de la sección eficaz para un único usuario y en el segundo se explicará cómo hacerlo para varios usuarios.

La cámara que se utiliza en este caso es la cámara User, ya que si se utilizase la cámara Depth se cogería también los objetos que rodean el entorno. Las variables globales necesarias para el cálculo de la sección eficaz son:

- Las variables de calibración de pantalla.
- PVector min: Se utiliza en caso de que solo queramos guardar una sección eficaz. En este vector se almacenarán la X mínima y la Y mínima del rectángulo que se va a generar.
- PVector max: Se utiliza en caso de que solo queramos guardar una sección eficaz. En este vector se almacenarán la X máxima y la Y máxima del rectángulo que se va a generar.
- Int [][] min: Se utiliza en caso de querer guardar varias secciones eficaces y tendrá un tamaño int[6][2] para poder guardar de 6 usuarios distintos la X mínima y la Y mínima.
- Int [][] max: Se utiliza en caso de querer guardar varias secciones eficaces y tendrá un tamaño int[6][2] para poder guardar de 6 usuarios distintos la X máxima y la Y máxima.

Como se puede observar la única diferencia en cuanto al número de usuarios es que para un usuario se utiliza un vector y para varios se utiliza un array, aunque si no se quisiera almacenar la información también se podría hacer para varios usuarios con un vector, uno a uno sobrescribiendo siempre el mismo. También se puede calcular con la librería BlobDetection, aunque este caso se explicará al final del apartado. Los pasos a seguir para calcular la sección eficaz para un usuario son:

1. Dar los valores iniciales en "void draw ()" a las variables para que aparezca el valor que aparezca se cojan los máximos y los mínimos, es decir, a los máximos se les da el valor de cero y a los mínimos se les da el valor de las dimensiones de la imagen.

```
min.x = width;
min.y = height;
max.x = 0;
max.y = 0;
```

2. Se hace un barrido a la imagen y si el pixel pertenece al usuario se calcula si sus coordenadas son más pequeñas que las mínimas o más grandes que las máximas para guardarlas si es así.

```
if (userMap[clickPosition] != 0) {
  cam.pixels[clickPosition] = color(255);
  if(x < min.x){
    min.x = x;
  }
  if(x > max.x){
    max.x = x;
  }
  if(y < min.y){
    min.y = y;
  }
  if(y > max.y){
    max.y = y;
  }
}
```

3. Con los puntos guardados en el punto 2 se dibujan líneas para ver visualmente el cercado del espacio.

```
stroke(255);
line(min.x,min.y,min.x,max.y);
line(min.x,min.y,max.x,min.y);
line(max.x,min.y,max.x,max.y);
```

Si en vez de querer calcular la sección eficaz de un usuario se quiere calculas las secciones eficaces de varios usuarios los pasos a seguir serían:

1. Dar los valores iniciales en "void draw ()" a todas las variables del array para que aparezca el valor que aparezca se cojan los máximos y los mínimos, es decir, a los máximos se les da el valor de cero y a los mínimos se les da el valor de las dimensiones de la imagen.

```
for(int s=0; s<6; s++){
 min[s][0] = width;
 min[s][1] = height;
 max[s][0] = 0;
 max[s][1] = 0;
}
```

Usuario por usuario por medio de un "for" utilizando "userList" se hace un barrido a la imagen y si el pixel pertenece al usuario se calcula si sus coordenadas son más pequeñas que las mínimas o más grandes que las máximas para guardarlas si es así.

```
if (kinect.getNumberOfUsers() > 0) {
 cam.loadPixels();
 for(int z = 0; z < userList.length; z++) {
  for(int h = 0; h < 480; h++){            //See all the pixels
   for(int w = 0; w < 640; w++){
     clickPosition = w + (h*640);         //We see which pixel we are working on
     if (userMap[clickPosition] != 0) {
         cam.pixels[clickPosition] = color(255);
         if(w < min[z][0]){
           min[z][0] = w;
         }
         if(w > max[z][0]){
           max[z][0] = w;
         }
         if(h < min[z][1]){
           min[z][1] = h;
         }
         if(h > max[z][1]){
           max[z][1] = h;
         }
       }
     }
  } //For end
```

2. Con los puntos guardados en el punto 2 se dibujan líneas para ver visualmente el cercado del espacio por cada usuario que se tiene, se hace al final del "for" y así se evita el tener que generar otro más.

```
stroke(255);
line(min[z][0],min[z][1],min[z][0],max[z][1]);
line(min[z][0],min[z][1],max[z][0],min[z][1]);
line(max[z][0],min[z][1],max[z][0],max[z][1]);
```

Como se puede ver la diferencia en el funcionamiento con uno o más usuarios siempre es igual, si es uno se utilizan vectores y si son más se utiliza un array para poder almacenar mayor cantidad de datos. Otro modo de calcular la sección eficaz sería por medio de la silueta, calculando esta y detectando que pixel de esta posee la X e Y mayor y menor, esto es lo que hace la librería BlobDetection, con la cual para calcularla se hace de la siguiente manera:

1. Llamar a la librería BlobDetection.

```
import blobDetection.*;
BlobDetection theBlobDetection;
```

2. Asignar la imagen que se va a trabajar a la librería y poner a esta las características de la imagen.

```
theBlobDetection = new BlobDetection(cam.width, cam.height);
theBlobDetection.setPosDiscrimination(false);
theBlobDetection.setThreshold(0.38f);
theBlobDetection.computeBlobs(cam.pixels);
```

3. Guardar en la imagen “cam” la captura de la cámara de la kinect, ya sea Depth o User, y colorear de un color los objetos y de otro el fondo de la captura.

```
cam.loadPixels();
for (int z=0; z<userList.length; z++){
  for(int h = 0; h < 480; h++){            //See all the pixels
     for(int w = 0; w < 640; w++){
       clickPosition = w + (h*640);         //We see which pixel we are working on
       if (userMap[clickPosition] != 0) {
           cam.pixels[clickPosition] = color(255);
         }
         else cam.pixels[clickPosition] = color(0);
       }
    }

}
cam.updatePixels();
```

4. Utilizar la función computeBlobs para calcular los puntos que pertenecen a la silueta.

```
theBlobDetection.computeBlobs(cam.pixels);
drawBlobsAndEdges(true, true);
```

5. Hacer un barrido de los pixeles de theBlobDetection, que son los pixeles de la imagen, para ver cuales pertenecen a la silueta o no.

```
for (int n=0 ; n<theBlobDetection.getBlobNb() ; n++)
{
  b=theBlobDetection.getBlob(n);
  if (b!=null)
  {
```

6. Calcular cuales con los puntos con la X y la Y mínima y máxima.

```
if (drawBlobs)
{
  strokeWeight(1);
  stroke(255, 0, 0);
  rect(
  b.xMin*width, b.yMin*height,
  b.w*width, b.h*height
    );
}
```

Finalmente ambos modos llevan a una misma solución, siendo más cómodo el uso de la librería BlobDetection para dibujar el rectángulo que representa la sección eficaz, pero siendo más cómodo el otro modo para trabajar con los datos para otras finalidades.

CÁLCULO DEL MOVIMIENTO EN EL ESCENARIO

En este trabajo respecto al movimiento se ha utilizado la cantidad de movimiento que hay en el escenario y no el flujo óptico, ya que este es la cantidad de movimiento que hay y la dirección de este. Para la cantidad de movimiento se puede utilizar tanto la cámara Depth como la cámara User ya que lo que se hace es un barrido de los pixeles de dos capturas y se comparan, mirando que pixeles han cambiado de estado, en la cámara User se verá el cambio de "0" a "1" y en la cámara Depth se mirará la distancia de este punto si ha variado en gran medida. Utilice la cámara que se utilice las variables globales que se utilizarán son:

- Las variables de calibración de pantalla.
- Int[] array1: Es un array de tamaño igual al número de pixeles que posee la pantalla de la kinect, 640x480=307200 pixeles. En este array se almacenará la imagen que se quiere guardar para compararla a la siguiente que capture la kinect. Se inicializa con todos sus valores a cero.
- Int[] array2: Es un array de tamaño igual al número de pixeles que posee la pantalla de la kinect, 640x480=307200 pixeles. En este array se almacenará la imagen que se acaba de capturar con la kinect. Se inicializa con todos sus valores a cero.
- Int num: Es una variable que se utiliza para la cuenta de pixeles que han cambiado de una imagen a otra.

El ejemplo que se va a utilizar para explicar cómo calcular la cantidad de movimiento se utiliza la cámara Depth. Una vez que se han declarado las variables y se han inicializado los array a cero los pasos a seguir son:

1. En "void setup ()" se inicializan los array a cero para que no de error durante el inicio.

```
for(int k=0; k<307200; k++){
  array1[k] = 0;
  array2[k] = 0;
}
```

2. Se hace un barrido de la imagen capturada por la kinect para en array2 poner un "1" si en el pixel hay algún cuerpo y un "0" si no lo hay.

```
int[] depthValues = kinect.depthMap();  //Save the Depth's values in an array
cam.loadPixels();
for(int x = 0; x < 640; x++){           //See all the pixels
  for(int y = 0; y < 480; y++){
    clickPosition = x + (y*640);        //We see which pixel we are working on
    clickedDepth = depthValues[clickPosition];    //See the pixel's value
    if (clickedDepth > 455){
      if (maxValue > clickedDepth){
        array2 [clickPosition] = 1;
        cam.pixels[ clickPosition] = color(0, 200, 0);
      }
      else array2 [clickPosition] = 0;
    }
  }
}
cam.updatePixels();
```

3. Se comparan array1 y array2 para ver cuántos pixeles han cambiado de "1" a "0" o viceversa y con esta cantidad saber cuánto movimiento ha habido.

```
for(int r=0; r<307200; r++){
  if(array1[r] != array2[r]){
    num++;
  }
}
num = int(num*reScale);
```

4. Se guarda la información de array2 en array1.

```
for(int r=0; r<307200; r++){
  array1[r] = array2[r];
}
```

Siguiendo estos pasos se consigue que continuamente se esté comparando la imagen actual de la kinect con la capturada anteriormente. Este método solo te permite saber el movimiento que ha habido lateralmente y no frontal mente, ya que detecta los cambios de píxeles de "0" a "1" y viceversa. Un modo de mejorarlo sería con la cámara depth comparando la distancia a la que se encuentra, aunque por el error de la cámara podría pensar que hay movimiento cuando el usuario se encuentra parado.

CÁLCULO DEL FLUJO ÓPTICO

Para el cálculo del flujo óptico se ha creado una clase llamada "Mov" con la cual se calcula todos los parámetros necesarios. Los ejemplos que se van a utilizar para la explicación son del escenario "partículas flotando", por lo que se enfocará también al movimiento de dichas partículas. Para calcular el flujo óptico se necesita detectar un punto que se mueve y compararlo con el mismo pero en su posición anterior para saber la distancia que les separa y la dirección.

Inicialmente el escenario se ha dividido en celdas creando una cuadrícula de todo el espacio. Para saber que fuerza se debe de ejercer sobre la partícula dependiendo del flujo óptico se pide dos parámetros, la celda en la que se encuentra y la distancia del punto que se está mirando respecto al que se ha guardado anteriormente, cada vez que se calcula el flujo este nuevo punto se almacena como "punto anterior".

```
a.x = max(1, a.x);      //take highest value, 1 or mouseX.
a.y = max(1, a.y);
float pointDx = a.x - oldBody[n][0];
float pointDy = a.y - oldBody[n][1];
int cellX = floor(a.x / cellWidth);
int cellY = floor(a.y / cellHeight);
oldBody[n][0] = a.x;
oldBody[n][1] = a.y;
```

Finalmente para el cálculo del flujo se utiliza la función "lerp" para interpolar linealmente ambos vectores (el punto actual y el anterior) y así tener el flujo óptico, que en el caso de "partículas flotando" marcará el movimiento de la partícula.

```
double dx = u[INDEX(cellX, cellY)];    //Retuns the cell postition as a pixel array
double dy = v[INDEX(cellX, cellY)];
cell = INDEX(cellX, cellY);
u[cell] = (vx != 0) ? PApplet.lerp((float) vx, (float) dx, 0.85f) : dx;
v[cell] = (vy != 0) ? PApplet.lerp((float) vy, (float) dy, 0.85f) : dy;
```

DIFUMINADO

En algunos escenarios se ha utilizado un difuminado para algunas formas como en los cubos que se mueven dependiendo de donde está el usuario o para algunos entornos como en el que parece niebla. Este difuminado se ha generado a partir de una imagen rectangular de un color a la cual se le ha ido dando a cada pixel una transparencia u otra. El código utilizado para este difuminado es:

```
PImage img;
img = createImage(50,50,ARGB);
for(int i = 0; i < square.pixels.length; i++) {
  float a = map(i, 0, square.pixels.length, 255, 0);
  img.pixels[i] = color(255, 255, 255, a);
}
```

En el código que se muestra en la imagen se siguen los siguientes pasos:

1. El código comienza creando una imagen, que en el ejemplo que se muestra es de 50x50 de tipo ARGB (una imagen RGB a la que se le puede añadir transparencia).
2. Se le aplica un barrido a todos los pixeles de la imagen, desde el pixel de la posición "0" hasta el pixel que se encuentra en el último lugar, la longitud de los pixeles de la imagen.
3. Se crea una variable tipo "float" (variable numérica con decimales) que será la transparencia que va a poseer el pixel. Esta variable coge un valor entre 255 y 0, siendo 255 si se encuentra en la posición "0" o siendo 0 si se encuentra en última posición. El código "map" permite dar valores intermedios entre los dos extremos dependiendo la posición que ocupe el pixel (i).
4. Finalmente se le da al pixel seleccionado un color con una transparencia "a".

PROYECCIÓN DE IMÁGENES

Como se pudo ver en el apartado anterior "Elementos estéticos" las imágenes pueden ayudar mucho en las proyecciones sobre un escenario, como es en el caso de las máscaras que se movían por el escenario flotando. Lo ideal a la hora de proyectar imágenes es que sean imágenes del tipo .png, ya que a este tipo de imágenes se les puede eliminar el fondo para que solo quede el objeto que aparece en la imagen. Para poder proyectar una imagen se debe de seguir una serie de pasos:

1. Guardar la imagen en la misma carpeta en la que se guarda el programa.
2. Declarar una variable del tipo PImage.
   ```
   PImage f1;
   ```
3. Cargar en dicha variable la imagen que hemos guardado y queremos utilizar.
   ```
   f1 = loadImage ("cara2SF.png");
   ```
4. Aplicar modificaciones sobre la imagen por medio de Scale: Aplica una escala determinada a la imagen. Se puede aplicar a la figura entera (scale(s)) o a los ejes por separado (scale(x,y,z)).
   ```
   scale(escala[w]);
   ```

5. Aplicar modificaciones sobre la imagen por medio de translate: Desplaza la imagen los pixeles que se quiera en los ejes con el código translate(x, y, z).

```
translate(0,0,pos[w][2]);
```

6. Aplicar modificaciones sobre la imagen por medio de ImageMode: Elige el modo en el que se va a proyectar la imagen respecto a coordenadas y dimensiones. Si no aparece la imagen se escribirá con el código image (imagen, coordenada x de la esquina superior izquierda, coordenada y de la esquina superior izquierda) con las dimensiones de la imagen. Si aparece imageMode (CENTER) la imagen se escribirá con el código image (coordenada x del centro de la imagen, coordenada y del centro de la imagen, ancho de la imagen, alto de la imagen). Si aparece imageMode (CORNER) la imagen se escribirá con el código image (coordenada x de la esquina superior izquierda, coordenada y de la esquina superior izquierda, ancho de la imagen, alto de la imagen). Si aparece imageMode (CENTER) la imagen se escribirá con el código image coordenada x de la esquina superior izquierda coordenada y de la esquina superior izquierda, coordenada x de la esquina inferior derecha, coordenada y de la esquina inferior derecha).

7. Mostrar la imagen dependiendo del modo que se haya puesto anteriormente.

```
scale(escala[w]);
translate(0,0,pos[w][2]);
imageMode(CENTER);
image(f1,pos[w][0],pos[w][1]);
```

CUBOS EN 3D

El primer paso para crear objetos en 3D es utilizar una librería que utilice 3D, estas son P3D u OPENGL. Para utilizar P3D solamente hay que llamarla en "setup" con el código "size (640, 480, P3D);", mientras que para utilizar OPENGL se debe de llamar primero y utilizarla en "setup" después:

```
import processing.opengl.*;
size(1366,768,OPENGL);
```

El ejemplo que se va a utilizar en este apartado es con un cubo. Para crear un objeto en 3D hay que crearlo y una vez creado con su centro en (0,0) se utiliza translate para moverlo en el espacio y rotateX, rotateY y rotateZ para rotar el cuerpo en los tres ejes. Los cubos del escenario "Cubos en movimiento al pasar" del apartado anterior "Elementos estéticos" se han creado de la siguiente manera:

```
pushMatrix();
rotateX(radians(ax));
rotateY(radians(-ay));
fill(0,100+r*25,0);
translate(lado*(r-q+6), -alto*r/12, -lado*6/10-lado*q);
box(lado,alto*r/6,lado);
popMatrix();
```

```
pushMatrix();
translate(100*x,100*y,0);
rotateX(radians(45+a[s]));
rotateY(radians(45+a[s]));
fill(175);
stroke(255);
box(lado);
popMatrix();
```

Como se puede observar no es como con los cuerpos en 2D, los cuales se crean con un tamaño y un origen determinado, estos se crean con un tamaño y a partir de de ahí se van moviendo hasta la posición deseada.

USO DE DOS CÁMARAS KINECT

Para poder utilizar dos cámaras kinect en un mismo programa es necesario tener las dos kinect conectadas a dos puertos USB distintos y a su fuente de alimentación. A la hora de escribir código la estructura es la misma cambiando el código, primero se debe de declarar que cámaras y que tipo de imagen se va a utilizar, luego hay que actualizarlas en "void draw()" y finalmente se debe de pedir los datos y se trabaja con estos.

Las partes de código que cambian son:

- A la hora de llamar a la librería en vez de llamar a la kinect, se llama a cada una de las cámaras que se van a utilizar, este punto es importante ya que cam1 y cam2 se utilizarán durante todo el programa:

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
```

```
import SimpleOpenNI.*;
SimpleOpenNI  cam1;
SimpleOpenNI  cam2;
```

- Para inicializar SimpleOpenNI se escribe un nuevo código, antes innecesario, "SimpleOpenNI.start();"

- Se crea una lista con las cámaras que se encuentran conectadas para después confirmar que hay dos cámaras y no solo una y se escribe el nombre de dicha cámara:

```
StrVector strList = new StrVector();
SimpleOpenNI.deviceNames(strList);
for(int i=0;i<strList.size();i++)
  println(i + ":" + strList.get(i));
```

- Se inicializan ambas cámaras y se verifican que estén funcionando con el siguiente código:

```
cam1 = new SimpleOpenNI(0,this,SimpleOpenNI.RUN_MODE_MULTI_THREADED);
cam2 = new SimpleOpenNI(1,this,SimpleOpenNI.RUN_MODE_MULTI_THREADED);
if(cam1.isInit() == false || cam2.isInit() == false)
{
   println("Verify that you have two connected cameras on two different usb-busses!");
   exit();
   return;
}
```

- A la hora de activár el tipo de cámaras que debe de utilizar cada kinect se tiene que indicar que cámara se activa. El modo de indicarlo es escribiendo, por ejemplo, "kinect.enableDepth();" el código "camX.enableDepth();". En este aspecto no cambia mucho el código, ya que hace referencia al nombre que se le haya dado a cada kinect al principio, el gran cambio es que tendremos que escribirlo dos veces, una por cada kinect.

- El resto de ordenes se escribe de manera similar a cuando tenemos una kinect únicamente, los dos únicos cambios es que se debe de escribir dos veces (una por cada kinect) y que se debe de utilizar el prefijo "camX." En vez de "kinect."

El resto de órdenes que posee la kinect se utilizan del mismo modo que con una kinect, siempre diferenciando entre los datos que nos da una cámara o la otra, ya que si se quieren los datos de la cámara depth obtendremos los datos de ambas o seleccionaremos que datos de que cámara queremos. Un ejemplo de programa con dos kinect conectadas es:

UNIÓN DE VARIOS ESCENARIOS

Para poder unir escenarios lo más importante es crear un buen control de estos y tener un buen orden, ya que como se va a mostrar a continuación son muchos datos y muchas funciones juntas que deben de coordinarse para trabajar bien. El programa utilizado para este apartado es el programa para el centro cultural explicado en el apartado anterior "Elementos estéticos". Este programa posee alrededor de 1769 líneas de código, por lo que el orden en funciones y pestañas es muy importante.

Para la unión de varios escenarios el programa final deberá de dividirse en varias partes, las cuales a su vez deberán de dividirse entre la cantidad de escenarios que se hayan utilizado en el programa para evitar liarse con tantos datos y tantas funciones. Después de haber decidido que escenarios se van a utilizar estos se separarán en función al control que se va a hacer con ellos, en el ejemplo se dividieron los escenarios en dos grupos, un grupo para una única persona y otro grupo para más de una persona. Estos escenarios y grupos se pondrán en orden de uso, lo cual facilitará el trabajo.

En la primera parte se declararán todas las librerías que se utilizan en los escenarios como Box2D, SimpleOpenNI, etc. y todas las variables dividiéndolas en grupos dependiendo de qué librería sean o de que escenario sean. Estas variables serán las que se utilicen para todo el programa, tanto control como escenarios que lo componen.

Un ejemplo sería así:

```
//Kinect
int clickedDepth,clickPosition;
int kinectWidth = 640;
int kinectHeight = 480;
int maxValue = 2500;
float reScale;

//BlobDetection
PImage cam, blobs;

//Sparkles
int[] array1SP = new int [307200];
int[] array2SP = new int [307200];
int numSP;
int countSP;
int actSP;

//Fog
PImage lightF;
PVector l1F = new PVector(0,0);
PVector l2F = new PVector(0,0);
PVector l0F = new PVector(0,0);
float[] scaleF = new float[3];
int[][] posF = new int [14][2];     //{x,y}
int[][] minF = new int[6][2];     //{person}{x,y}
int[][] maxF = new int[6][2];
```

Después en "void setup ()" se darán los valores deseados a las variables declaradas anteriormente, dividiendo estos también por escenario o librería. Por ejemplo:

```
//Box2D
box2d = new PBox2D(this);
box2d.createWorld();
box2d.setGravity(0, 0);

//BlobDetection
theBlobDetection = new BlobDetection(cam.width, cam.height);
theBlobDetection.setPosDiscrimination(false);
theBlobDetection.setThreshold(0.38f);
theBlobDetection.computeBlobs(cam.pixels);

//Program
 for(int o=0; o<11; o++){
  end[o] = 0;
  endFinished[o] = 0;
}

//Sparkles
numSP = 0;
countSP = 0;
actSP = 0;
for(int o=0; o<307200; o++){
  array1SP[o] = 0;
  array2SP[o] = 0;
}
```

El siguiente paso es en “void draw ()” llamar a una función que será la que controla el paso de un escenario a otro dependiendo de los usuarios que se encuentren frente al proyector y crear un “switch()” que tendrá una función por cada escenario y dependerá de una variable llamada “scene” para ir cambiando con el paso del tiempo como se muestra en la imagen de la derecha. Cada función dependerá de unos valores procedentes de la kinect que se obtendrán antes de llamar a las funciones, por lo que se les dará como dato dependiendo de cuál necesiten, así evitamos el copiar código igual en todas las funciones.

```
people = userList.length;
controlP();
countScene ++;

switch(scene){
  case 0:
    sparkles(depthValues);
    break;
  case 1:
    fog(userList, depthValues);
    break;
  case 2:
    whitePoints(depthValues);
    break;
```

Dentro de la función de control se genera el orden por el que van a salir cada uno de los escenarios siendo este 0>1>2>3>4>0 y 5>6>7>8>9>10>5. Además tendrá dos variables (“people” y ”peopleBefore”) con las que localizará cuando ha entrado una persona y debe de cambiar a escenarios de más de una persona o cuando se ha salido alguien y puede volver a escenarios de una persona.

Otra función importante es “ending”, la cual se llama cada vez que finaliza el tiempo de cada escenario o cuando el control avisa de que hay que cambiar por número de personas. En esta función se avisa de que se está cambiando de escenario y se hace una cuenta atrás de 5 a 0, apareciendo así el nuevo escenario.

Cuando se cambia de escenario también se llama a la función “reset”, que reinicia los valores de todas las variables de los escenarios para que estos empiecen desde cero, así se evitan problemas como que en el escenario de la nieve la nieve empiece a caer y no aparezca en la mitad de la proyección.

```
////////////////////////////////////////  ENDING  /////

void ending(){

  countEnding++;
  println(countEnding);

  translate(0,0,ZNumbers);
  background(0);

  String s = "cambiando de escenario";
  fill(175);
  noStroke();
  textSize(32);
  text(s,width/7,50);

  for(int u=0; u<5; u++){
    if(u*60 == countEnding){ZNumbers = -60;}
  }

  if(countEnding  < 60){
    textSize(256);
    text("5",width/3,2*height/3);
    ZNumbers++;
  }
  else if(countEnding < 120){
    textSize(256);
    text("4",width/3,2*height/3);
    ZNumbers++;
  }
```

Cada función comienza preguntándose con un "if" si está por debajo del tiempo límite para cada función para poder continuar, si no es así la función llamará a "ending" para cambiar de escenario como se muestra a continuación:

```
////////////////////////////////////  WHITEPOINTS  ////////////,

void whitePoints(int[] depthValues){

  if( countScene > time){
    ending();
  }
  else if(end[scene]==1 || control == 1){
    ending();
  }
  else{

    background(0);
    IntVector userList = new IntVector();
    kinect.getUsers(userList);

    for (int z=0; z<userList.size(); z++){
      int userId = userList.get(z);           //getting user data
      kinect.getCoM(userId, position);
      kinect.convertRealWorldToProjective(position, position);

      jointPos.x = position.x*reScale;
      jointPos.y = position.y*reScale;
```

FORMAS CREADAS

Durante la realización de este trabajo se han ido realizando distintos cuerpos y figuras de diversas formas y colores. Todos ellos poseen una lógica y unos cálculos que permiten tener como resultado la forma deseada. En este apartado vamos a ver las distintas formas y cómo se han realizado en el código del programa, los cuerpos más representativos son:

COPO DE NIEVE

El copo de nieve se ha creado por medio de un código que funciona con variables, de tal manera que variando dichas variables puedas mover y cambiar de tamaño al copo. Las variables utilizadas son:

- Float x: Coordenada X del centro del copo de nieve.
- Float y: Coordenada Y del centro del copo de nieve.
- Float lado: Es la longitud que se quiere para el lado del copo de nieve.

El copo de nieve se va formando ramificación a ramificación, saliendo del centro ocho de estas. De cada una de estas ramificaciones salen otras dos, teniendo entre ellas un ángulo de 45º.

El código y el resultado es el siguiente:

```
noFill();
 stroke(255);
 line(x,y-lado/2,x,y+lado/2);
 line(x,y-lado/3,x+lado/6,y-lado/2);
 line(x,y-lado/3,x-lado/6,y-lado/2);
 line(x,y+lado/3,x+lado/6,y+lado/2);
 line(x,y+lado/3,x-lado/6,y+lado/2);
 line(x-lado/2,y,x+lado/2,y);
 line(x-lado/3,y,x-lado/2,y+lado/6);
 line(x-lado/3,y,x-lado/2,y-lado/6);
 line(x+lado/3,y,x+lado/2,y-lado/6);
 line(x+lado/3,y,x+lado/2,y+lado/6);
 pushMatrix();
 noFill();
 stroke(255);
 translate(x,y,0);
 rotateZ(radians(45));
 line(0,-lado/2,0,lado/2);
 line(0,-lado/3,lado/6,-lado/2);
 line(0,-lado/3,-lado/6,-lado/2);
 line(0,lado/3,lado/6,lado/2);
 line(0,lado/3,-lado/6,lado/2);
 rotateZ(radians(90));
 line(0,-lado/2,0,lado/2);
 line(0,-lado/2,0,lado/2);
 line(0,-lado/3,lado/6,-lado/2);
 line(0,-lado/3,-lado/6,-lado/2);
 line(0,lado/3,lado/6,lado/2);
 line(0,lado/3,-lado/6,lado/2);
 popMatrix();
```

Como se observa se divide el lado en 3 partes iguales y la segunda ramificación se inicia a 2/3 de la longitud del lado del centro. Se utiliza pushMatrix () y popMatrix () para que las rotaciones que se efectúan no afecten al resto del código, ya que si no se aplicarían a todo el código restante.

ÁRCOLES Y CESPED

Estas formas representan unos árboles de invierno blancos con césped que van creciendo desde el suelo hasta que están completos. Para generar este "crecimiento" se crean variables que guardarán la posición inicial y final tanto de los árboles como del césped. La dinámica de este programa es ir sumando unos valores a las variables que se utilizan para crear las líneas hasta que se complete el árbol, en este apartado enseñaremos el código final del árbol y no todo el código del proceso son sus sumas. Las variables globales que se crean para el césped son:

- Float [][] coordI: Indica las coordenadas iniciales {césped}{x, y}. La cantidad de césped utilizada en los escenarios es de 400.
- Float [][] coordF: Indica las coordenadas finales {césped}{x, y}. La cantidad de césped utilizada en los escenarios es de 400 y se le suele dar una altura de 15.
- Float [][] coordP: Indica las coordenadas del punto en el que nos encontramos {césped}{x, y}. la cantidad de césped utilizada en los escenarios es de 400.
- Float [] aleatorio: Es un número entre el -1 y el 1 e indica la inclinación que va a tener el césped.

Las variables globales que se crean para los árboles son:

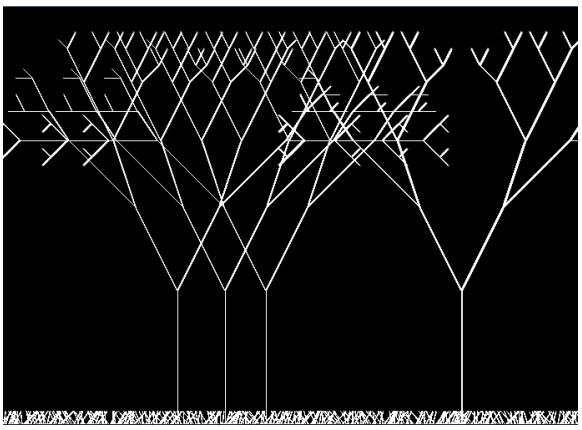
- Int [] momento: indica en que parte del árbol nos encontramos. Es un array de 6 ya que se generan en los escenarios 6 árboles, aunque esto se puede variar.
- Int [][] x: Es el valor que posee la coordenada X en cada rama de cada árbol y es al que se le irá sumando valores para que el árbol vaya creciendo. Tiene un tamaño de {6 árboles} {63 ramas}.
- Int [][] y: Es el valor que posee la coordenada Y en cada rama de cada árbol y es al que se le irá sumando valores para que el árbol vaya creciendo. Tiene un tamaño de {6 árboles} {63 ramas}.
- Int [][] cuentax: Es un array con 4 cuentas que controlan el crecimiento de los árboles. Su tamaño es {6 árboles} {4 cuentas}.
- Int [] sx1, sx2, sx3: Tienen un tamaño de 6 por los 6 árboles y son los que almacenan cuanto hay que sumar a X para que el árbol crezca. Hay tres ya que dependiendo que de rama se trate deberá crecer con una velocidad u otra.
- Int [] sy1, sy2, sy3: Tienen un tamaño de 6 por los 6 árboles y son los que almacenan cuanto hay que sumar a Y para que el árbol crezca. Hay tres ya que dependiendo que de rama se trate deberá crecer con una velocidad u otra.

Los árboles se van creando por partes, la primera parte está formada por una única rama a la que se le va sumando una cantidad en su eje X e Y para que vaya creciendo hasta que llega a una altura de 160. En ese momento se pasa a la parte dos, la cual tiene el doble de ramas que la anterior, es decir, 2. Esta parte va creciendo hasta que alcanza una altura de 100, que sumados a los 160 de la primera parte hacen un árbol con una altura de 260. En ese momento se pasa a la parte dos, la cual tiene el doble de ramas que la anterior, es decir, 4. Así sigue creciendo continuamente hasta llegar a la parte 6 con 32 ramas y una altura de 470.

A modo de ejemplo mostraremos la parte 3 del árbol, donde se puede observar que empieza dibujando la parte 1 y 2 y luego pasa a sumar en la parte 3 y a dibujarla. La imagen pertenece al escenario completo.

```
case 2:
  line(x[c][0],y[c][0],x[c][1],y[c][1]);
  line(x[c][1],y[c][1],x[c][3],y[c][3]);
  line(x[c][1],y[c][1],x[c][2],y[c][2]);

  sy1[c]++;
  sx2[c]++;
  sy2[c]++;
  if(cuentax[c][0] > 2){
    sx1[c]++;
    cuentax[c][0] = 0;
  }
  x[c][5] = x[c][3] + sx1[c];
  y[c][5] = y[c][3] - sy1[c];
  x[c][4] = x[c][3] + sx2[c];
  y[c][4] = y[c][3] - sy2[c];
  x[c][6] = x[c][2] - sx1[c];
  y[c][6] = y[c][2] - sy1[c];
  x[c][7] = x[c][2] - sx2[c];
  y[c][7] = y[c][2] - sy2[c];
  fill(255);
  stroke(255);
  line(x[c][3],y[c][3],x[c][4],y[c][4]);
  line(x[c][3],y[c][3],x[c][5],y[c][5]);
  line(x[c][2],y[c][2],x[c][6],y[c][6]);
  line(x[c][2],y[c][2],x[c][7],y[c][7]);
```



```
  if(sy1[c] == 80){
    sx1[c] = 0;
    sy1[c] = 0;
    sx2[c] = 0;
    sy2[c] = 0;
    momento[c]++;
  }
break;
```

ROBOT

Otra de las formas realizadas durante este trabajo ha sido el robot. Su función ha sido sobre todo el aprender a programar con la librería Box2D, ya que es la que se utiliza para crear dicho objeto, y para mostrar al aula de las artes que se puede crear cualquier cosa por medio del código. Para la creación solo ha hecho falta la librería Box2D, que ha sido la librería de física utilizada en este trabajo.

La clase "robot" se divide en tres partes con una función cada una:

1. Características: En esta parte se pueden añadir características como la densidad, si es un cuerpo estático o dinámico, la posición de inicio, si comienza con alguna velocidad inicial, etc.

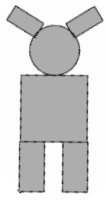
```
BodyDef bd = new BodyDef();
bd.type = BodyType.DYNAMIC;
bd.position.set(box2d.coordPixelsToWorld(x,y));
body = box2d.createBody(bd);
body.setLinearVelocity(new Vec2(random(-5, 5), random(2, 5)));
body.setAngularVelocity(random(-5, 5));
```

2. Partes: Se crean todas las partes del cuerpo con sus formas dentro del mundo de Box2D, estas formas no serán las que se vean. En el ejemplo siguiente se puede observar una pierna del robot, el cual es un rectángulo.

```
PolygonShape pd = new PolygonShape();
float box2dW2 = box2d.scalarPixelsToWorld(3);
float box2dH2 = box2d.scalarPixelsToWorld(6);
pd.setAsBox(box2dW2, box2dH2);
```

3. Display: Se dibujan las formas creadas en el punto anterior como si el centro fuese el punto (0,0), ya que después ya se moverá dicho cuerpo al punto deseado. Estas formas son las que sí se ven.

```
void display() {
    Vec2 pos = box2d.getBodyPixelCoord(body);
    float a = body.getAngle();
    rectMode(CENTER);
    pushMatrix();
    translate(pos.x,pos.y);
    rotate(-a);
    fill(175);
    stroke(0);
    rect(0,0,80,80);
    ellipse(0,-70,30*2,30*2);
    rect(26,70,30,60);
    rect(-26,70,30,60);
    rotate(-45);
    rect(70,-85,20,40);
    rotate(90);
    rect(-70,-85,20,40);
    popMatrix();
  }
```



HILOS QUE UNEN

Los hilos del escenario "Hilos que unen usuarios" se han creado por medio de curvas con unos puntos que dependen de variables como la distancia ente usuarios, la altura del centro de masas, etc. para así poder hacer este escenario con los usuarios que sean y sin importar el lugar en el que se encuentren. También se les ha añadido un movimiento aleatorio para crear más dinamismo en el escenario.

A la hora de dibujar los hilos se han ido haciendo uno a uno, utilizando solo 4 valores para la X y otros 4 para la Y se han creado los 8 puntos necesarios para dibujarlos. El punto 0 y 8 serán aquellos que no se ven en la curva y el resto serán los que sí se ven, siendo el número 4 el del centro. Al igual que para crear los hilos que pasan por arriba y los hilos que pasan por debajo se ha utilizado el mismo código pero invirtiendo el eje Y.

Se ha utilizado para dibujar los hilos las siguientes variables:

- Int [] x: Es un array de amplitud 4 para guardar las coordenadas del eje X de los puntos necesarios para generar los hilos.
- Int [] y: Es un array de amplitud 4 para guardar las coordenadas del eje Y de los puntos necesarios para generar los hilos.
- Float [][] jointPos: Array de 15x2 que almacena los puntos de los centros de masas para crear los hilos entre estos, ya que es necesario un punto inicial y un punto final de la curva.
- Int num: Es en número del hilo que se está dibujando, ya que para cada hilo se utilizan unos valores en los puntos u otros. El hilo "0" es el que se encuentra en el centro y el número "8" es el superior y el inferior, los hilos de arriba y de debajo utilizan los mismos valores pero con la Y invertida.

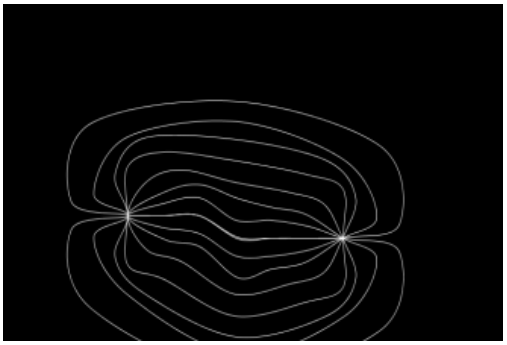
Para dibujar los hilos se puede dividir en dos partes:

1. El cálculo de los puntos utilizando para cada hilo (num) por medio de un "for":

```
case 1:
  x[0] = -q/6 + int(random(-5,5));
  x[1] = q/6 + int(random(-5,5));
  x[2] = q/3 + int(random(-5,5));
  x[3] = q/2 + int(random(-5,5));
  y[0] = -15*num + int(random(-5,5));
  y[1] = 15*num + int(random(-5,5));
  y[2] = 20*num + int(random(-5,5));
  y[3] = 25*num + int(random(-5,5));
  break;
```

2. El dibujo de las curvas utilizando todos los puntos calculados anteriormente:

```
noFill();
stroke(255);
beginShape();
curveVertex(x1+x0,y1+y0);
curveVertex(x1,y1);
curveVertex(x1+x2,y1+y2);
curveVertex(x1+x3,y1+y3);
curveVertex(x1+x4,y1+y4);
curveVertex(x7-x3,y7+y3);
curveVertex(x7-x2,y7+y2);
curveVertex(x7,y7);
curveVertex(x7+x0,y7+y0);
endShape();
noFill();
stroke(255);
beginShape();
curveVertex(x1+x0,y1-y0);
curveVertex(x1,y1);
curveVertex(x1+x2,y1-y2);
curveVertex(x1+x3,y1-y3);
curveVertex(x1+x4,y1-y4);
curveVertex(x7-x3,y7-y3);
curveVertex(x7-x2,y7-y2);
curveVertex(x7,y7);
curveVertex(x7+x0,y7-y0);
endShape();
```



PUESTA EN ESCENA

El resultado escénico de este proyecto ha sido programado dentro del Óxido Fest, cuyas bases son las sinergias que la Danza establece con otras disciplinas y/o fuentes de creación. Es por ello que la estructura de ÓXIDO FEST -tanto en su conceptualización como en la propia organización del evento- hace alusión a la estructura molecular del óxido de hierro de Lewis, de manera que tras la muestra de los trabajos el espectador pueda por sí mismo interpretar la amplia amalgama de posibilidades de interacción que existen entre la danza y la tecnología, la robótica, el vídeo, el cine, la fotografía, etc. El óxido es por otro lado un concepto que remite al desgaste que los metales padecen con su exposición a la intemperie, lo que entronca directamente con otro de los leitmotif del festival: la calle (como metáfora de espacios no convencionales de representación teatral) y el lugar del celebración "El sitio de mi recreo", Centro Juvenil de la Comunidad de Madrid que se caracteriza fuertemente por su edificio de paredes oxidadas.

La primera de las decisiones de la puesta en escena consistió en la elección del espacio donde presentar el proyecto. Las alternativas brindadas por el festival eran amplias, por lo que pudimos visitar todas las instalaciones del centro reduciendo finalmente nuestro interés a cuatro espacios:

- Una sala polivalente con espejos.
- El escenario del teatro.
- Dos fachadas exteriores.

Finalmente, las fachadas exteriores quedaron descartadas por solicitud del festival, que tenía dos grandes problemas, dar cobertura a esta localización por cuestiones de producción (pantalla gigante, música, permiso de uso público...) y la iluminación que habría a la hora asignada para la actuación, las 17:00 del domingo día 22 de Junio. Decidimos descartar el teatro y apostar por la sala polivalente por la presencia del espejo, que permite al intérprete ver su interacción con la proyección sin necesidad de emplear un monitor, lo que sería muy útil para la tercera parte de la presentación. Los handicaps y soluciones que tenía esta elección son:

- Demasiada luz natural y dificultad para hacer un oscuro total que favorezca la proyección. Finalmente conseguimos tapar las ventanas con un sistema temporal que impedía el paso de luz de la calle.

- La superficie de proyección era de color oscuro y reflectante. El color nos pareció una característica interesante, ya que estábamos programados en ese peculiar edificio y no en otro, podríamos aprovechar esta característica e integrarla en nuestra muestra cómo demostración de la capacidad de diálogo con el entorno. Pero finalmente, al tratarse de una superficie satinada decidimos forrar la superficie de proyección para evitar reflejos que pudieran ser incómodos para el público asistente.

La estructura que decidimos para compartir el proyecto con el público tras el estudio de la programación general del festival que nos acogía y de la población de la zona fue la siguiente:

1. Charla de presentación del proyecto para explicar verbalmente en qué consiste el trabajo que hemos realizado.
2. Muestra de un espectáculo breve para demostrar las posibilidades de la relación entre la danza y las nuevas tecnologías.
3. Apertura del dispositivo a que personas del público puedan probar los distintos escenarios que hemos desarrollado, para que a través del juego y de dinámicas participativas puedan completar el comprendimiento de la interacción con la proyección a través del sensor.

La dramaturgia del espectáculo, la selección musical y la interpretación fueron aportaciones del Aula de Danza de la Universidad, que tuvieron por título "El cuerpo aumentado".

La estructura del espectáculo se divide en cuatro partes con cinco escenarios en total. La estructura es la siguiente:

- La primera parte estará compuesta únicamente de voz que leerá un texto a la vez que el bailarín baila en un único escenario. El escenario utilizado es “Fogonazos con movimiento”.
- La segunda parte no posee ni música ni voz. Es un momento de corte en el que se utiliza el silencio junto con un nuevo escenario. El escenario utilizado es “Líneas en cruz”.
- La tercera parte posee una pieza de musical dividida en dos partes, una con instrumentos de viento y otra de instrumentos de cuerda. En esta parte se utilizan dos escenarios. Dichos escenarios son “Fondo de círculos blancos” y “Fuerzas en partículas”.
- La última parte del espectáculo está formada por una pieza que une música y voz, finalizando así con una unión de las características de las partes anteriores. El escenario utilizado para esta parte es “Luz que se desvanece v2.0”

Ambos textos aparecen en “Anexo 4: Texto del Óxido Fest”, el cartel del festival en el “Anexo 1: Cartel Óxido Fest” y la entrada en “Anexo 6: Entrada Óxido Fest” al final del documento.

Finalmente el espectáculo se realizó dentro del escenario utilizando una de sus paredes laterales, ya que en este escenario no había tanta luz y era más recogido, ya que el público estuvo a lo largo del escenario. Se comenzó con una explicación de que se había hecho y el porqué, explicando también todos los proyectos que está llevando a cabo el aula de danza con la universidad. Después se llevó a cabo el espectáculo que tuvo una duración de 8 minutos y 50 segundos y de la cual se pueden observar imágenes en “Anexo 5: Imágenes Óxido Fest”, terminando con una explicación de lo mostrado y con un apartado de preguntas.

CONCLUSIONES

Inicialmente buscaba un proyecto del que se pudiese dar un uso de cara al exterior, algo que sirviese al resto de personas, algo que no se realizase y más tarde se quedase en una estantería o en un disco duro guardado con una nota asignada. Desde mi punto de vista ese primer objetivo personal se ha cumplido, ya que gracias al aula de las artes de la UC3M he conseguido realizar un trabajo que ha podido disfrutar la gente al ver nuestras actuaciones y que espero puedan seguir disfrutando en un futuro.

Personalmente creo que ha sido un trabajo complejo y largo, tanto en el campo de la programación por hacerlo en "Processing", un programa poco documentado que posee muchos ejemplos y ayudas con versiones muy anteriores a las que inicialmente se utilizan, como en el campo del diálogo tecnología-arte, ya que son mundos muy distintos que en unión pueden sacar cosas geniales y mientras que uno busca la perfección visual, que capte la cámara todo bien, el otro busca el transmitir sin palabras, solo con gestos, el crear un mundo irreal para sentirse dentro de este.

Otro punto que ha ralentizado el trabajo ha sido el estar compuesto de tanta gente de tantos lugares diferentes, es decir, el hecho de haber un profesor de universidad, gente del aula de las artes con sus actuaciones y demás actos, un organizador del festival y yo, un estudiante de universidad que trabaja por la tarde. Esta ralentización ha sido a la hora de poder quedar y poder realizar ensayos donde se buscaban cosas y luego aparecían otras nuevas.

Personalmente he aprendido mucho tanto de la programación en Processing, de la cual no sabía nada cuando escogí el trabajo a principio de curso, como en el mundo del arte, esa manera de ver las cosas, de cómo sacar a partir de una luz el significado de que esa luz es la vida que se puede hacer más grande o que puede desvanecerse; cómo en el trabajo en equipos de gente tan diferente pero que se une para hacer cosas como esta y debe de estar bien comunicado y trabajar a la par, ha sido un trabajo enriquecedor.

Finalmente agradecer la ayuda de mi tutor Javier F. Gorostiza del departamento de automática, de Alfredo Miralles del aula de danza de la Universidad Carlos III de Madrid y de toda la gente que me ha ayudado a lo largo de este proceso.

TRABAJOS FUTUROS

Un modo de mejorar este trabajo de cara al futuro es el incorporarle la nueva cámara de Microsoft, la kinect 2, y utilizarla en lugar de la primera versión de la kinect. Esto es debido a que la imagen de la nueva cámara es mucho mejor que la de su antecesora, mejorando en precisión, sensibilidad y profundidad. Es tres veces más sensible, pudiendo reconocer partes tan pequeñas como las arrugas de la ropa y el campo de visión es 60º más amplio, pudiendo concentrar a seis personas en la pantalla. Y finalmente respecto a sonido es mucho mejor, ya que te puede reconocer la voz aunque haya mucho ruido en la sala.

Aquí se puede observar la diferencia de imagen de la kinect 1 (imagen de la izquierda) y la kinect 2 (imagen de la derecha):

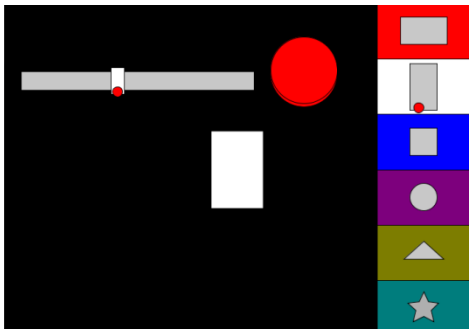
Otra mejora a añadir a este trabajo sería añadirle reconocimiento de audio, pudiendo utilizar la voz para controlar los cambios de escenario o utilizando música para interactuar con el escenario que se esté proyectando, haciendo que la interactuación se lleve a cabo con el ritmo de la música también.

En relación con el audio un interesante trabajo sería que la música que va sonando varíe dependiendo de los movimientos del usuario, pudiendo ser más rápida con movimientos rápidos o más lenta con movimientos lentos. Este trabajo sería complejo pero daría muy buenos resultados a la hora de crear un espectáculo.

Con la plataforma “Processing” y la kinect otro posible trabajo sería darle un enfoque más pedagógico creando juegos para niños en los que se utilice una pared como pantalla y el cuerpo del usuario como mando para controlar el juego. Se podría hacer juegos de colores, de construcción, de rapidez, etc.

TRABAJOS PARALELOS

Paralelamente al trabajo realizado con el aula de las artes de la universidad Carlos III de Madrid se ha ido realizando un juego de construcción que también aprovecha las cualidades de la kinect junto a las proyecciones que se crean con un proyector. El juego es un juego sencillo de construcción con 6 figuras diferentes que se pueden mover y agrandar o empequeñecer como el usuario desee.



En la imagen de la derecha se puede observar como es el juego. Para estas imágenes se ha creado una curcunferenceia roja en cada mano para saber que esta realizando el usuario, ya que las imágenes estar sacadas del ordenador.

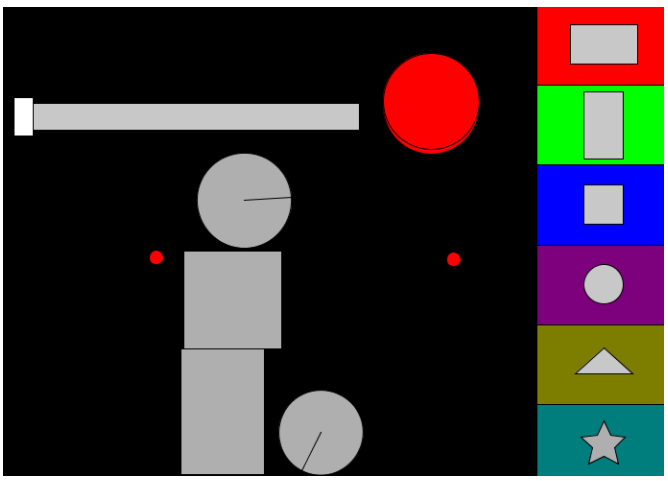
A la derecha se ha dividido una columna en seis rectángulos y en cada unno de estos aparece una figura diferente que poseerá fondo blanco cuando se seleccione la figura, como se puede observar en la imagen con el rectángulo seleccionado.

El botón rojo creado sirve para pasar de un paso a otro y la barra de la izquierda agranda o decrece la figura seleccionada, siendo lo más pequeña posible al situarse el rectángulo a la izquierda y lo más grande posible al situarse el rectángulo a la derecha.

El guión a seguir para jugar es:

1. Seleccionar una de las figuras que aparecen a la derecha con una de las manos.
2. Seleccionar el tamaño de la figura elegida por medio de la barra que se encuentra en la parte superior izquierda moviendo el rectángulo que se encuentra en esta con la mano.
3. Pulsar el botón rojo para que se bloquee la selección de figura y de tamaño.
4. Pasar una mano por encima de la figura creada para empezar a moverla.
5. Mover la figura a la zona deseada y pulsar el botón rojo para crearla.

Un ejemplo del juego funcionando es la imagen de debajo, en la cual se ha creado un rectángulo, un cuadrado y dos círculos.



BIBLIOGRAFÍA

1. Como hacer un estado del arte en el enlace http://formandoinvestigadores-gft.blogspot.com.es/2011/01/estado-del-arte.html
2. Libro "Making things see: 3D vision with kinect, Processing and Arduino", febrero 2012, Greg Borenstein.
3. Algorithms for Visual Design Using the Processing Language. Published May 2009, Kostas Terzidis.
4. Processing 2: Creative Coding Hotshot, mayo 2013, Nikolaus Gradwohl.
5. Processing: www.processing.org
6. OpenNI SDK v2.1: http://www.openni.org/openni-sdk/.
7. NiTE v2.2.0.11: http://www.openni.org/files/nite/
8. SimpleOpenNI v1.96: http://code.google.com/p/simple-openni
9. Processing v2.1: http://processing.org/download/?processing
10. GitHub: https://github.com
11. Box2D: http://box2d.org
12. Giselle: http://www.ciudaddeladanza.com/bibliodanza/argumentos-de-ballet/giselle.html
13. Giselle: http://teatres.gva.es/danza/ficha-espectaculo-danza/espectaculo_324/giselle
14. Sinestesia: http://naukas.com/2011/12/30/que-es-la-sinestesia-que-sabemos-sobre-ella-que-nos-queda-por-saber/
15. Sinestesia: http://www.elconfidencial.com/sociedad/2011/sinestesia-musica-colores-20110129-74091.html
16. Sinestesia: documento "BELLA MOLINA MARIA LARA. PSICOLOGIA TEORICA 2005. PROFESOR EMILIO GOMEZ MILAN"
17. Sinestesia: http://blogdenortiz.blogspot.com/2011/06/la-sinestesia-en-el-arte-y-en-la-musica.html
18. Metáfora cuerpo-Proyección: http://www.redalyc.org/articulo.oa?id=82200912
19. Kinect 2: http://es.gizmodo.com/descubre-de-cerca-como-es-la-kinect-2-y-el-nuevo-mando-509248086
20. Estado del arte: http://www.errequeerredanza.net/?p=523&lang=es
21. Estado del arte: http://avatarsolo.blogspot.com.es/
22. Estado del arte: http://www.madrid.org/fo/2012/es/fichas/danza/stocos.html

ANEXOS

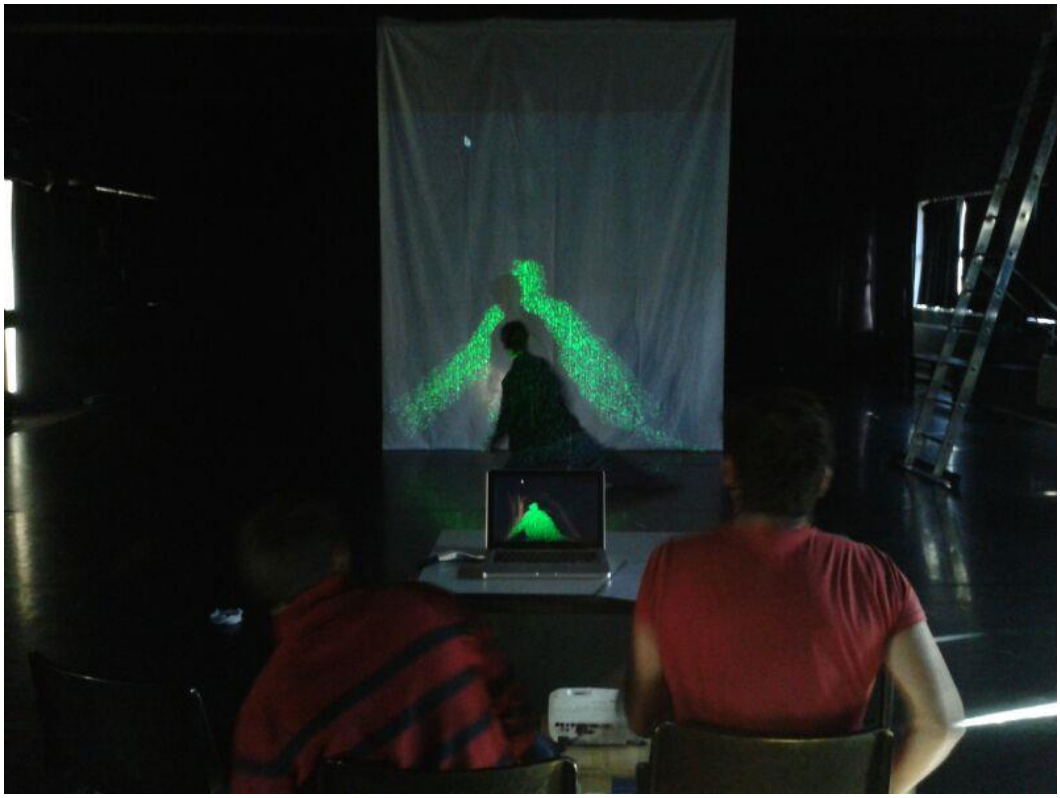
ANEXO 1: CARTEL DEL ÓXIDO FEST

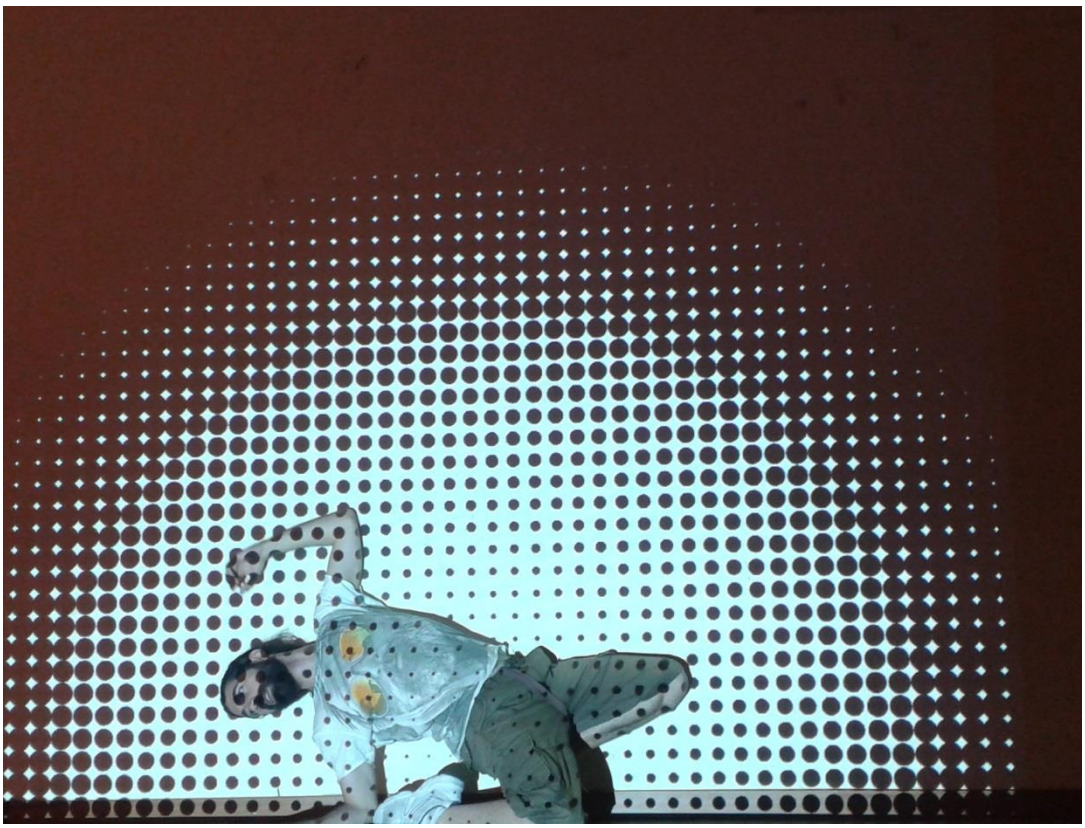
ANEXO 2: IMÁGENES DE "EL SITIO DE MI RECREO"

Fachada del centro juvenil "El sitio de mi recreo"

Tejado del centro juvenil "El sitio de mi recreo"

ANEXO 3: IMÁGENES DE LOS ENSAYOS





ANEXO 4: TEXTO DEL ÓXIDO FEST

En la actuación que se realiza en el Óxido Fest aparecen dos textos que serán leídos durante la actuación. Estos textos son:

Texto 1:

Antes de que cumplas 11 años
te darás cuenta de que existe la maldad
Y tendrás que decidir entre combatirla o vivir tu vida al margen.

Una de las dos opciones te convertirá en un héroe, para algunos. La otra, en un héroe, para otros. Porque no hay buenos ni malos; sólo hay héroes que hacen lo que pueden. No es sencillo ser un superhéroe hoy en día.

Es posible que todavía no lo hayas descubierto. Que pienses que eres como todos los demás, pero
tu superpoder te aguarda. El mundo te necesita. La humanidad necesita ser salvada. Necesitamos salvarnos de ti. Si no tienes un superpoder eres una amenaza.

Cuando cumplas 11 años habrás aprendido que la maldad no estaba fuera.
La única lucha con sentido es la que mantienes contigo mismo.
Es... como sobrevivir. Nunca puedes ganar. La victoria siempre es momentánea. La única victoria es seguir luchando. Seguir corriendo el riesgo de perder.
La recompensa es escasa. Como si el premio de los valientes fuera seguir teniendo miedo.

Yo también tuve miedo cuando supe todo esto.
A los 11 años no se puede ser más un niño. Es la única edad a la que pueden hacerse las cosas importantes de la vida.

Soy valiente. Pero... ¿qué pasaría si mi único superpoder es ser consciente de mi fragilidad?
¿Quién me necesitaría? ¿Alguien gritaría mi nombre en búsqueda de auxilio?
QUIZÁ NO SOY DE ESOS SUPERHÉROES QUE SALVAN A LA GENTE
IGUAL MI SUPERPODER SÓLO SIRVE PARA SALVARME A MÍ MISMO.

Texto 2:

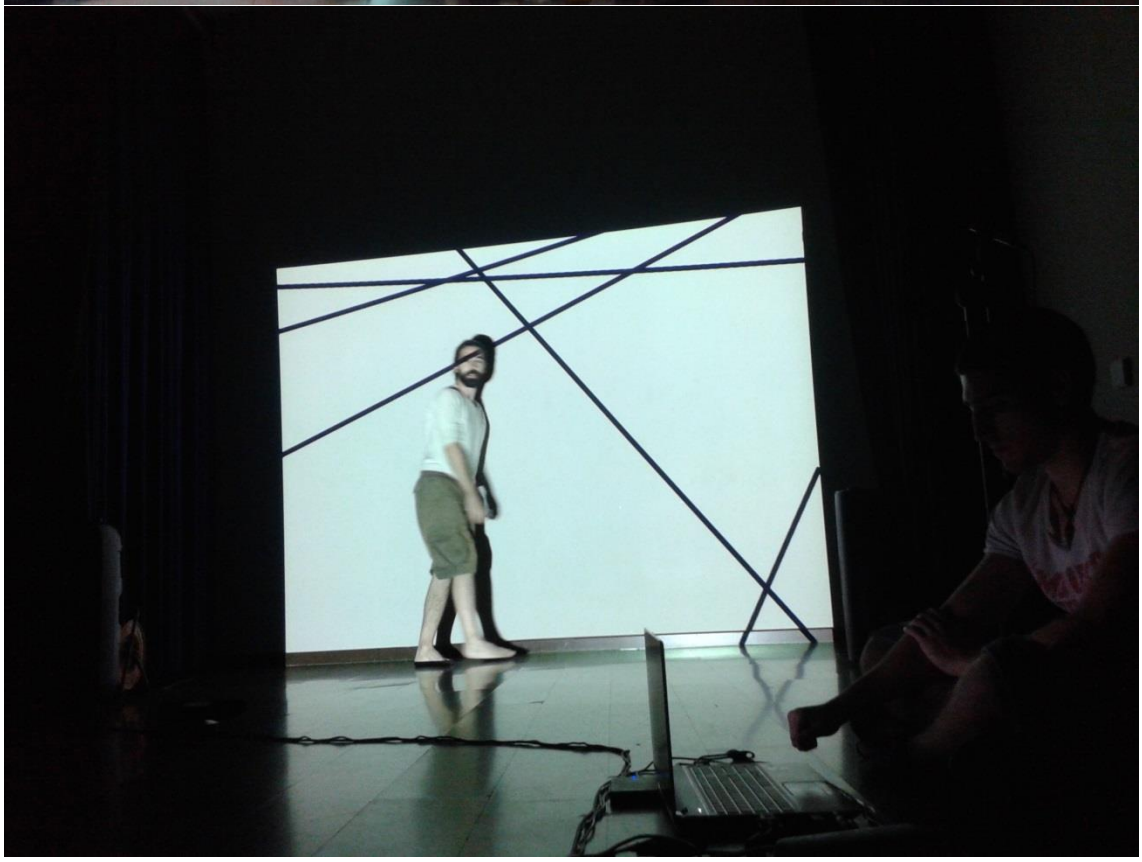
Ya no podrás decir que no lo sabías.
Ya no conserves más esa inocencia.

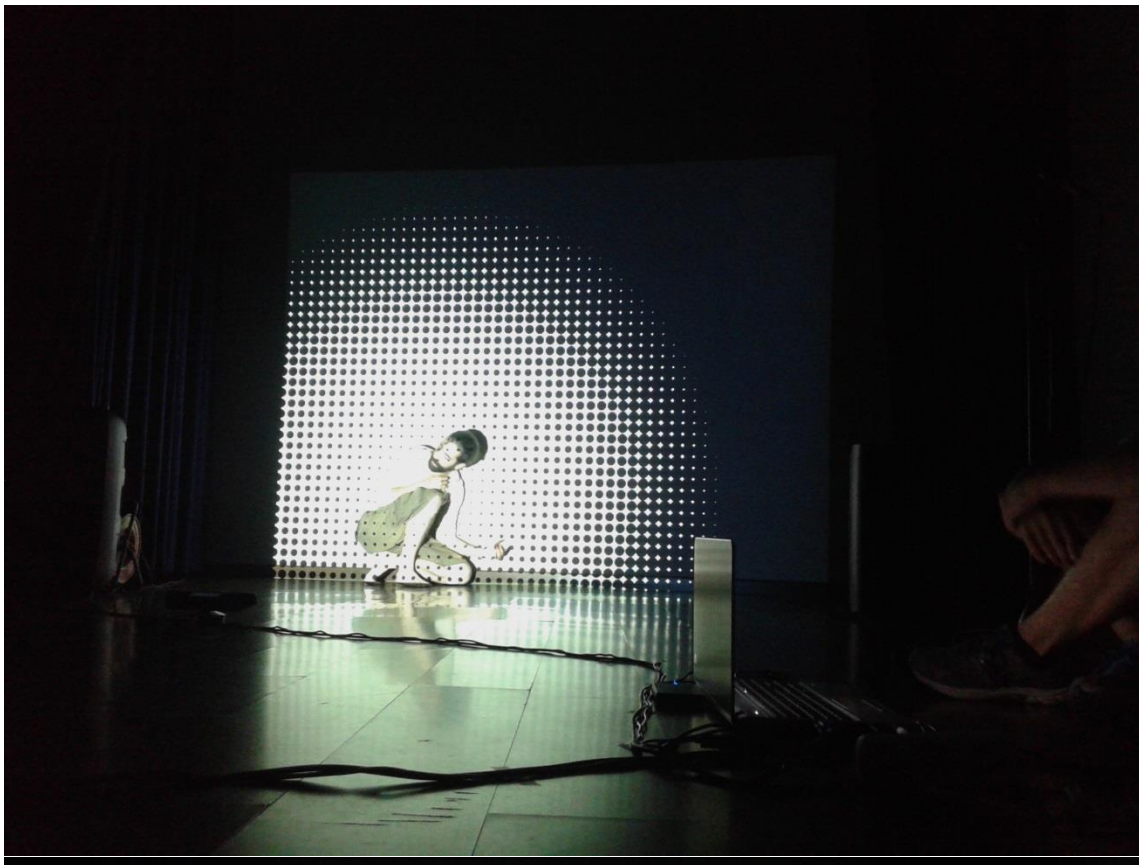
Por favor, no te creas que tus manos terminan en tus dedos.
Siente como tu piel es permeable
y el aire a tu alrededor te roza por dentro.
No pienses que tu pierna es tu articulación más larga,
acuérdate de tus ojos, que llegan hasta el horizonte.

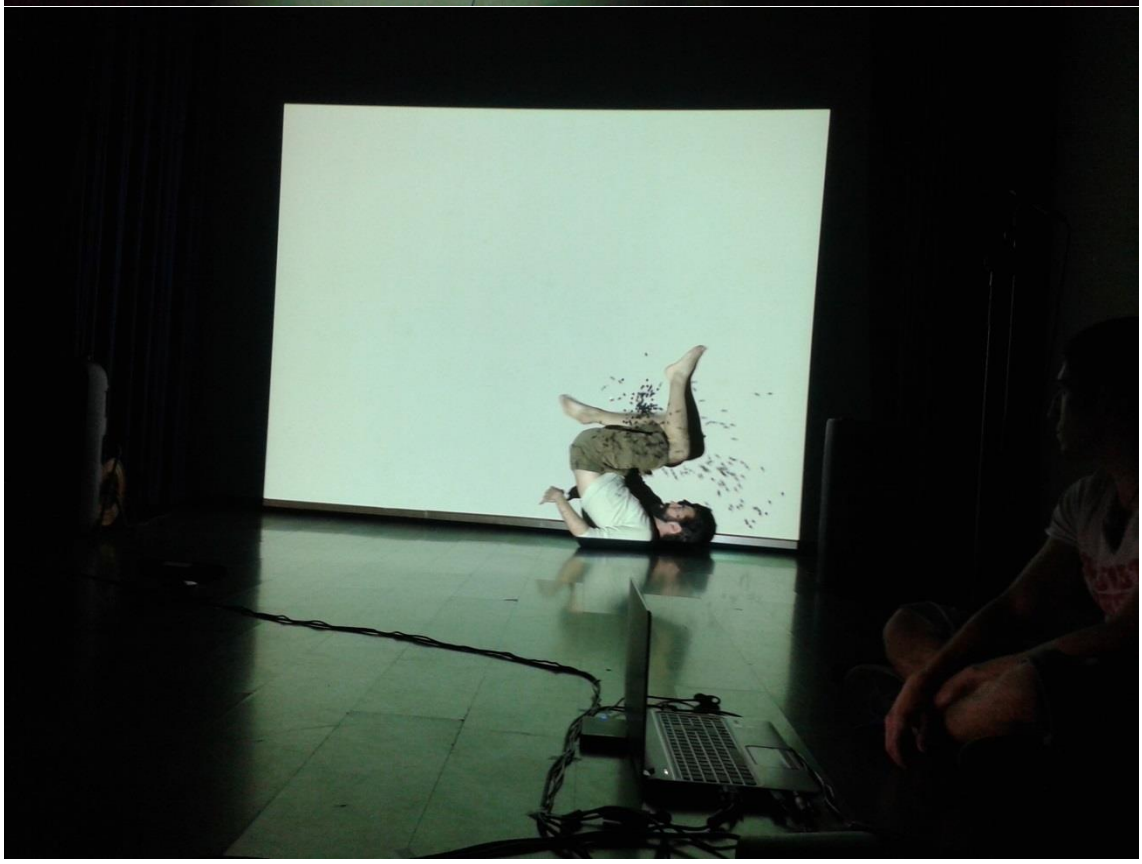
Por favor, vive en tus cuerpos sutiles,
descubre su ingravidez, que también te pertenece.
Recuerda cuando sabías volar.
Por favor, no pienses en tus pies como el final de tu cuerpo.
Siente la repercusión de tu presencia en lo profundo de la tierra.
Sé tus alas y tus raíces.

Porque si la meta no está fuera,
si el camino es hacia dentro,
qué más da perderse en el espacio-tiempo.
Y si mi cuerpo no me limita y
soy uno con el universo,
cómo sé quién soy, a dónde voy y de dónde vengo.

ANEXO 5: IMÁGENES ÓXIDO FEST







ANEXO 6: ENTRADA ÓXIDO FEST

ANEXO 7: CÓDIGO DEL PROGRAMA DEL ÓXIDO FEST

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
import pbox2d.*;
import org.jbox2d.collision.shapes.*;
import org.jbox2d.common.*;
import org.jbox2d.dynamics.*;
PBox2D box2d;
import blobDetection.*;
BlobDetection theBlobDetection;
import java.awt.Polygon;
import java.util.Collections;
import processing.opengl.*;

//Program
int control = 0;
int time = 800000000;
int scene = 0;
int people = 5;
int peopleBefore = 5 ;
int futureScene = 0;
int countEnding = 0;
int countScene = 0;
int ZNumbers = -60;
int[] end = new int[11];
int[] endFinished = new int[11];
int[] userMap ;
int[] userList ;
int[] depthValues ;
```

```
PVector position = new PVector(0,0,0);
PVector jointPos = new PVector(0,0,0);

//Kinect
int clickedDepth,clickPosition;
int kinectWidth = 640;
int kinectHeight = 480;
int maxValue = 2600;
float reScale;
PImage cam;

//Sparkles
int[] array1SP = new int [307200];
int[] array2SP = new int [307200];
int numSP;
int countSP;
int actSP;

//whitePoints
float max_distance;

//Explosion
int[][] nowEX = new int[7][2];
int[][] beforeEX = new int[7][2];
int[][] minEX = new int[6][2];    //{person}{x,y}
int[][] maxEX = new int[6][2];
int [] pixelEX;
int allParticlesEX = 250;
ArrayList<Boundary> boundaries;
ArrayList<Particle> particles;
```

```
//ClosingCircle
int comienzoCC;
int radioCC;
int cuentaPixelsCC;
int PixelsCC = 0;

//AvoidThreads
int[][] linesTH;              //{line}{y0,y1}
int[] actLinesTH;
int[][] minTH = new int[6][2];    //{person}{x,y}
int[][] maxTH = new int[6][2];
int cTH;
int nLinesTH = 150;

////////////////////////////////// SETUP ////////////////////////////////////

void setup(){
 size(1024,768,OPENGL);
 background(0);
 reScale = (float) width / kinectWidth;
 cam = createImage(640,480,RGB);

 //Kinect
 kinect = new SimpleOpenNI(this);
 kinect.enableDepth();
 kinect.enableUser();
 kinect.setMirror(true);

 if (kinect.isInit()==false){
```

```
 println("Nada");
 exit();
 return;
}

//Box2D
box2d = new PBox2D(this);
box2d.createWorld();
box2d.setGravity(0, 0);

//Program
 for(int o=0; o<11; o++){
  end[o] = 0;
  endFinished[o] = 0;
}

//Sparkles
numSP = 0;
countSP = 0;
actSP = 0;
for(int o=0; o<307200; o++){
  array1SP[o] = 0;
  array2SP[o] = 0;
}

//whitePoints
max_distance = dist(0, 0, width, height);

//Explosion
particles = new ArrayList<Particle>();
```

```
boundaries = new ArrayList<Boundary>();
boundaries.add(new Boundary(width/2,0,width,10));
boundaries.add(new Boundary(0,height/2,10,height));
boundaries.add(new Boundary(width/2,height-5,width,10));
boundaries.add(new Boundary(width-5,height/2,10,height));
pixelEX = new int[width*height];
for(int s=0; s<6; s++){
  minEX[s][0] = width;
  minEX[s][1] = height;
  maxEX[s][0] = 0;
  maxEX[s][1] = 0;
}

for(int r=0; r<allParticlesEX; r++){
 Particle p = new Particle(random(width),random(height),2);
 particles.add(p);
}

for(int q=0; q<pixelEX.length; q++){
 pixelEX[q] = 0;
}

for(int u=0; u<4; u++){
 beforeEX[u][0] =0;
 beforeEX[u][1] = 0;
 nowEX[u][0] = 0;
 nowEX[u][1] = 0;
}

//closingCircle
```

```
  PixelsCC = height*width;
  radioCC = int(width*2);
  comienzoCC = 0;
  cuentaPixelsCC = 100000;

}

////////////////////////////////////// DRAW //////////////////////////////////////

void draw(){
  println(countScene);
  println("Scene: "+ scene);
  println("People: "+ people);

  box2d.step();
  kinect.update();
  userMap = kinect.userMap();
  userList = kinect.getUsers();
  depthValues = kinect.depthMap();

  if(userList.length>0){
    people = userList.length;
  }
  controlP();
  //peopleBefore = people;
  countScene ++;

  switch(scene){
    case 0:
      sparkles(depthValues);
```

```
    break;
   case 1:
    threads(userList);
    break;
   case 2:
    whitePoints(depthValues);
    break;
   case 3:
    explosion(userList, userMap);
    break;
   case 4:
    closingCircle(depthValues);
    break;
   default:
    background(0,255,0);
    break;
  }
}
/////////////////////////////////// BOUNDARY ////////////////////////////////
class Boundary {
  // A boundary is a simple rectangle with x,y,width,and height
  float x;
  float y;
  float w;
  float h;

  // But we also have to make a body for box2d to know about it
  Body b;

  Boundary(float x_,float y_, float w_, float h_) {
```

```
  x = x_;
  y = y_;
  w = w_;
  h = h_;

  // Define the polygon
  PolygonShape sd = new PolygonShape();
  // Figure out the box2d coordinates
  float box2dW = box2d.scalarPixelsToWorld(w/2);
  float box2dH = box2d.scalarPixelsToWorld(h/2);
  // We're just a box
  sd.setAsBox(box2dW, box2dH);


  // Create the body
  BodyDef bd = new BodyDef();
  bd.type = BodyType.STATIC;
  bd.position.set(box2d.coordPixelsToWorld(x,y));
  b = box2d.createBody(bd);

  // Attached the shape to the body using a Fixture
  b.createFixture(sd,1);
}

// Draw the boundary, if it were at an angle we'd have to do something fancier
void display() {
  fill(0);
  stroke(0);
  rectMode(CENTER);
  rect(x,y,w,h);
```

```
 }
}
//////////////////////////////// FUNCTIONS ////////////////////////////////
//////////////////////////////// SPARKLES ////////////////////////////////
void sparkles(int[] depthValues){
 if( countScene > time){
  ending();
 }
 else if(end[scene]==1 || control == 1){
  ending();
 }
 else{
  fill(0,0,0,30);
  rect(0,0,width*2,height*2);
  cam.loadPixels();
  for(int x = 0; x < kinectWidth; x++){         //See all the pixels
   for(int y = 0; y < kinectHeight; y++){
    clickPosition = x + (y*kinectWidth);       //We see which pixel we are working on
    clickedDepth = depthValues[clickPosition];    //See the pixel's value
    if (clickedDepth > 455){
     if (maxValue > clickedDepth){
      array2SP [clickPosition] = 1;
      cam.pixels[ clickPosition] = color(175);
     }
     else array2SP [clickPosition] = 0;
    }
   }
  }
  cam.updatePixels();
  for(int r=0; r<307200; r++){
```

```
  if(array1SP[r] != array2SP[r]){
    numSP++;
  }
 }
println("flujo optico: " + numSP);
println("act: " + actSP);
if(numSP > 6000 && actSP==0){
 actSP=1;
}
if(actSP==1 && countSP<4){
 fill(255);
 noStroke();
 rect(0,0,width,height);
 stroke(0,0,50);
 strokeWeight(10);
 line(int(random(0,width)),0,0,int(random(0,height)));
 line(int(random(0,width)),height,width,int(random(0,height)));
 line(int(random(0,width)),0,int(random(0,width)),height);
 line(0,int(random(0,height)),width,int(random(0,height)));
 line(int(random(0,width)),0,0,int(random(0,height)));
 countSP++;
}

if(countSP == 4){
 countSP=0;
 actSP=0;
}
for(int r=0; r<307200; r++){
 array1SP[r] = array2SP[r];
}
```

```
    numSP = 0;
  }
}
////////////////////////////// WHITEPOINTS ////////////////////////////////
void whitePoints(int[] depthValues){
  if( countScene > time){
    ending();
  }
  else if(end[scene]==1 || control == 1){
    ending();
  }
  else{
    background(255);
    IntVector userList = new IntVector();
    kinect.getUsers(userList);
    for (int z=0; z<userList.size(); z++){
      int userId = userList.get(z);          //getting user data
      kinect.getCoM(userId, position);
      kinect.convertRealWorldToProjective(position, position);
      jointPos.x = position.x*reScale;
      jointPos.y = position.y*reScale;
      jointPos.x = width-jointPos.x;
      for(int i = 0; i <= width; i += 20) {
        for(int j = 0; j <= height; j += 20) {
          float size = dist(jointPos.x, jointPos.y, i, j);
          size = size/max_distance * 66;
          fill(0);
          noStroke();
          ellipse(i, j, size, size);
        }
```

```
    }
   }
  }
}

//////////////////////////////// EXPLOSION ////////////////////////////////
void explosion(int[] userList,int[] userMap){
 if( countScene > time){
  ending();
 }
 else if(end[scene]==1 || control == 1){
  ending();
 }
 else{
  background(255);
  int totalEX = 0;
  for(int s=0; s<6; s++){
    minEX[s][0] = width;
    minEX[s][1] = height;
    maxEX[s][0] = 0;
    maxEX[s][1] = 0;
  }
  for (int z=0; z<userList.length; z++){
   int userId = userList[z]; //getting user data
   kinect.getCoM(userId, position);
   kinect.convertRealWorldToProjective(position, position);
   nowEX[z][0] = int(position.x*reScale);
   nowEX[z][1] = int(position.y*reScale);
   nowEX[z][0] = width - nowEX[z][0];
   for(int h = 0; h < kinectHeight; h++){        //See all the pixels
```

```
    for(int w = 0; w < kinectWidth; w++){
     clickPosition = w + (h*kinectWidth);       //We see which pixel we are working on
     if (userMap[clickPosition] != 0) {
      pixelEX[clickPosition] = 1;
      if(w < minEX[z][0]){
        minEX[z][0] = w;
       }
       if(w > maxEX[z][0]){
        maxEX[z][0] = w;
       }
       if(h < minEX[z][1]){
        minEX[z][1] = h;
       }
       if(h > maxEX[z][1]+5){
        maxEX[z][1] = h;
       }
     }
    }
  }
}
int aty=userList.length;
println("USERLIST: "+aty);
println("TOTAL: "+totalEX);
float d = maxEX[0][0]-minEX[0][0];
float n = maxEX[0][1]-minEX[0][1];
println("B-N: "+d+" ; "+n);
if (d<300) {
 int count=0;
 int s=0;
 for(int user=0; user<7; user++){
```

```
 if (nowEX[user][0]!=0 && nowEX[user][1]!=0){
  totalEX++;
 }
}
for (Particle b: particles) {
 switch(totalEX){
  case 1:
   for(int user=0; user<7; user++){
    if (nowEX[user][0]!=0 && nowEX[user][1]!=0){
     b.attract(nowEX[user][0],nowEX[user][1]);
    }
   }
   break;
  case 2:
   s = int(allParticlesEX/2);
   if(count<s){
    b.attract(nowEX[2][0],nowEX[2][1]);
   }
   else{
    b.attract(nowEX[1][0],nowEX[1][1]);
   }
   break;
  case 3:
   s = int(allParticlesEX/3);
   if(count<s){
    b.attract(nowEX[3][0],nowEX[3][1]);
   }
   else if(count<2*s){
    b.attract(nowEX[1][0],nowEX[1][1]);
   }
```

```
        else{
          b.attract(nowEX[2][0],nowEX[2][1]);
        }
        break;
      case 4:
        s = int(allParticlesEX/4);
        if(count<s){
          b.attract(nowEX[4][0],nowEX[4][1]);
        }
        else if(count<2*s){
          b.attract(nowEX[1][0],nowEX[1][1]);
        }
        else if(count<3*s){
          b.attract(nowEX[2][0],nowEX[2][1]);
        }
        else{
          b.attract(nowEX[3][0],nowEX[3][1]);
        }
        break;
    }
    count++;
  }
}
else{
  int r = 0;
  for (Particle b: particles) {
    r++;
    int x = int(r/65);
    switch(x){
      case 0:
```

```
        b.moveAway(random(0,width),0);
        break;
      case 1:
        b.moveAway(random(0,width),height);
        break;
      case 2:
        b.moveAway(0,random(0,height));
        break;
      case 3:
        b.moveAway(width,random(0,height));
        break;
    }
  }
}
for (Boundary wall: boundaries) {
  wall.display();
}
for (Particle b: particles) {
  b.display();
}
for (int i = particles.size()-1; i >= 0; i--) {
  Particle b = particles.get(i);
  if (b.done()) {
    particles.remove(i);
  }
}
for(int u=0; u<4; u++){
  beforeEX[u][0] = nowEX[u][1];
  beforeEX[u][0] = nowEX[u][1];
}
```

```
    for(int q=0; q<pixelEX.length; q++){
      pixelEX[q] = 0;
    }
  }
}
////////////////////////////// CLOSING CIRCLE ////////////////////////////////

void closingCircle(int[] depthValues){
  if( countScene > time){
    ending();
  }
  else if(end[scene]==1 || control == 1){
    ending();
  }
  else{
    fill(0);
    rect(0,0,width*3,height*3);
    IntVector userList = new IntVector();
    kinect.getUsers(userList);
    cam.loadPixels();
    for(int x = 0; x < 640; x++){          //See all the pixels
      for(int y = 0; y < 480; y++){
        clickPosition = x + (y*640);       //We see which pixel we are working on
        clickedDepth = depthValues[clickPosition];    //See the pixel's value
        if (clickedDepth > 455){
          if (maxValue > clickedDepth){
            cuentaPixelsCC++;
            cam.pixels[ clickPosition] = color(0);
          }
        }
```

```
 }
}
cam.updatePixels();
for (int z=0; z<userList.size(); z++){
 int userId = userList.get(z);          //getting user data
 kinect.getCoM(userId, position);
 kinect.convertRealWorldToProjective(position, position);
 jointPos.x = position.x*reScale;
 jointPos.y = position.y*reScale;
 jointPos.x = width-jointPos.x;
}
if (cuentaPixelsCC < 35000)  {
 comienzoCC = 1;
}
if (comienzoCC == 1){
 radioCC-=15;
 if(radioCC < 30){
  comienzoCC = 0;
  radioCC = int(width*2);
 }
 fill(255);
 ellipse(jointPos.x,jointPos.y,radioCC,radioCC);
}
else{
 fill(255);
 ellipse(width/2,height-height/20,radioCC,radioCC);
}
println("Comienzo: " + comienzoCC);
println("Pixels: " + cuentaPixelsCC);
cuentaPixelsCC = 0;
```

```
  /* translate(0, (height-kinectHeight*reScale)/2);
   scale(reScale);
   image(cam,0,0);*/
   }
}

////////////////////////////// STRANDS ////////////////////////////////////

void threads(int[] userList){
 if( countScene > time){
  ending();
 }
 else if(end[scene]==1 || control == 1){
  ending();
 }
 else{
  fill(0,0,0,35);
  rect(0,0,2*width,2*height);
  for (int i=0; i<userList.length; i++){
   int userId = userList[i]; //getting user data
   kinect.getCoM(userId, position);
   kinect.convertRealWorldToProjective(position, position);
   jointPos.x = position.x*reScale;
   jointPos.y = position.y*reScale;
   jointPos.x = width-jointPos.x;
   for(int n=0; n<15; n++){
    PVector v = new PVector();
    v.x = int(random(jointPos.x-50,jointPos.x+50));
    v.y = int(random(jointPos.x-50,jointPos.x+50));
    PVector h = new PVector();
```

```
      h.x = int(random(jointPos.y-50,jointPos.y+50));
      h.y = int(random(jointPos.y-50,jointPos.y+50));
      stroke(255);
      strokeWeight(1);
      line(v.x,0,v.y,height);
      line(0,h.x,width,h.y);
    }
   }
  }
}
////////////////////////////// GENERAL FUNCTIONS ////////////////////////////

////////////////////////////////  CONTROLP ////////////////////////////////
void controlP(){
 if(control == 0){
  switch(scene){
    case 0:
      futureScene = 1;
      break;
    case 1:
      futureScene = 2;
      break;
    case 2:
      futureScene = 3;
      break;
    case 3:
      futureScene = 4;
      break;
    case 4:
      futureScene = 5;
```

```
      break;
    case 5:
      futureScene = 6;
      break;
    case 6:
      futureScene = 7;
      break;
    case 7:
      futureScene = 8;
      break;
    case 8:
      futureScene = 9;
      break;
    case 9:
      futureScene = 10;
      break;
    case 10:
      futureScene = 0;
      break;
   }
  }
}
////////////////////////////////// ENDING ////////////////////////////////////
void ending(){
  countEnding++;
  println(countEnding);
  background(0);
  end[scene] = 0;
  scene = futureScene;
  futureScene = 0;
```

```
  countEnding = 0;
  countScene = 0;
  control = 0;
}
////////////////////////////////// RESET //////////////////////////////////

void reset(){
  //Sparkles
  numSP = 0;
  countSP = 0;
  actSP = 0;
  for(int o=0; o<307200; o++){
    array1SP[o] = 0;
    array2SP[o] = 0;
  }
  //Explosion
  particles = new ArrayList<Particle>();
  boundaries = new ArrayList<Boundary>();
  for(int s=0; s<6; s++){
     minEX[s][0] = width;
     minEX[s][1] = height;
     maxEX[s][0] = 0;
     maxEX[s][1] = 0;
  }
  for(int r=0; r<allParticlesEX; r++){
    Particle p = new Particle(random(width),random(height),2);
    particles.add(p);
  }
  for(int q=0; q<pixelEX.length; q++){
    pixelEX[q] = 0;
```

```
  }
  for(int u=0; u<4; u++){
   beforeEX[u][0] =0;
   beforeEX[u][1] = 0;
   nowEX[u][0] = 0;
   nowEX[u][1] = 0;
  }
  //closingCircle
  PixelsCC = height*width;
  radioCC = int(width*2);
  comienzoCC = 0;
  cuentaPixelsCC = 100000;
}
//////////////////////////////// MAXVALUE ////////////////////////////////
void keyPressed(){
  println("MaxValue: " + maxValue);
  switch(key)
  {
  case 'q':
   maxValue+=100;
   break;
  case 'a':
   maxValue-=100;
   break;
  case 'r':
   scene++;
   println(scene);
   reset();
   break;
  case 'f':
```

```
    scene--;
    reset();
    println(scene);
    break;
  case 'l':
    kinect.setMirror(!kinect.mirror());
    break;
  case 'm':
    countScene = time-10;
  }
}
////////////////////////////////// PARTICLE //////////////////////////////////
class Particle {
  // We need to keep track of a Body and a radius
  Body body;
  float r;
  color col;
  Particle(float x, float y, float r_) {
    r = r_;
    // This function puts the particle in the Box2d world
    makeBody(x, y, r);
    body.setUserData(this);
    col = color(0);
  }
  // This function removes the particle from the box2d world
  void killBody() {
    box2d.destroyBody(body);
  }
  float getPositionX(){
    Vec2 pos = box2d.getBodyPixelCoord(body);
```

```
  return pos.x;
 }
 float getPositionY(){
  Vec2 pos = box2d.getBodyPixelCoord(body);
  return pos.y;
 }
  void attract(float x,float y) {
  // From BoxWrap2D example
  Vec2 worldTarget = box2d.coordPixelsToWorld(x,y);
  Vec2 bodyVec = body.getWorldCenter();
  // First find the vector going from this body to the specified point
  worldTarget.subLocal(bodyVec);
  // Then, scale the vector to the specified force
  worldTarget.normalize();
  worldTarget.mulLocal((float) 5);
  // Now apply it to the body's center of mass.
  body.applyForce(worldTarget, bodyVec);
 }
 void moveAway (float x,float y) {
  // From BoxWrap2D example
  Vec2 worldTarget = box2d.coordPixelsToWorld(x,y);
  Vec2 bodyVec = body.getWorldCenter();
  // First find the vector going from this body to the specified point
  worldTarget.subLocal(bodyVec);
  // Then, scale the vector to the specified force
  worldTarget.normalize();
  worldTarget.mulLocal((float) 50);
  // Now apply it to the body's center of mass.
  body.setLinearVelocity(worldTarget);
 }
```

```
// Is the particle ready for deletion?
boolean done() {
  // Let's find the screen position of the particle
  Vec2 pos = box2d.getBodyPixelCoord(body);
  // Is it off the bottom of the screen?
  if (pos.y > height+r*2) {
    killBody();
    return true;
  }
  return false;
}
void display() {
  // We look at each body and get its screen position
  Vec2 pos = box2d.getBodyPixelCoord(body);
  // Get its angle of rotation
  float a = body.getAngle();
  pushMatrix();
  translate(pos.x, pos.y);
  rotate(a);
  fill(col);
  stroke(0);
  strokeWeight(1);
  ellipse(0, 0, r*2, r*2);
  // Let's add a line so we can see the rotation
  line(-r, 0, r, 0);
  popMatrix();
}
// Here's our function that adds the particle to the Box2D world
void makeBody(float x, float y, float r) {
  // Define a body
```

```
    BodyDef bd = new BodyDef();
    // Set its position
    bd.position = box2d.coordPixelsToWorld(x, y);
    bd.type = BodyType.DYNAMIC;
    body = box2d.createBody(bd);
    // Make the body's shape a circle
    CircleShape cs = new CircleShape();
    cs.m_radius = box2d.scalarPixelsToWorld(r);
    FixtureDef fd = new FixtureDef();
    fd.shape = cs;
    // Parameters that affect physics
    fd.density = 1;
    fd.friction = 0.01;
    fd.restitution = 0.3;
    // Attach fixture to body
    body.createFixture(fd);
    body.setAngularVelocity(random(-10, 10));
  }
}
```